Generación de Lenguaje Natural a partir de Grafos Semánticos

Trabajo de Fin de Máster
Curso 2015–2016

Autor
Adrián Rabadán Jurado

Director
Gonzalo Méndez Pozo
Raquel Hervás Ballesteros

Máster en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Documento maquetado con TEXIS v.1.0.

Este documento está preparado para ser imprimido a doble cara.

Generación de Lenguaje Natural a partir de Grafos Semánticos

Trabajo de Fin de Máster en Ingeniería Informática
Departamento de Ingeniería del Software e Inteligencia Artificial

Autor
Adrián Rabadán Jurado

Director
Gonzalo Méndez Pozo
Raquel Hervás Ballesteros

Convocatoria: *Septiembre 2016*
Calificación:

Máster en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

26 de septiembre de 2016

Autorización de difusión

El abajo firmante, matriculado en el Máster en Ingeniería Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "Generación de Lenguaje Natural a partir de Grafos Semánticos", realizado durante el curso académico 2015-2016 bajo la dirección de Gonzalo Méndez Pozo y Raquel Hervás Ballesteros en el Departamento de Ingeniería de Software e Inteligencia Artificial (ISIA), y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Adrián Rabadán Jurado

26 de septiembre de 2016

Agradecimientos

Quiero dar las gracias en primer lugar a todos los miembros de NIL, y especialmente a Gonzalo y a Raquel, por haberme dado la oportunidad de poder trabajar con ellos. Me he sentido muy a gusto y he podido descubrir el mundo laboral, que me ha hecho crecer como persona y como ingeniero.

También quiero agradecerle a mi familia todo el apoyo y el cariño que me han dado durante mi estancia en la facultad y el haberme aconsejado y guiado en la vida.

Me alegra mucho haber tenido buenos ratos con los amigos de siempre y con los de informática para desestresarme. Y agradezco haber conocido a mis compañeros de trabajo del *aula 16*, con los que me lo he pasado en grande riéndonos, jugando y comiendo.

Durante este año han ocurrido muchas cosas, y cada año mejor estoy contigo. Me encanta pasar mi tiempo libre en tu compañía, Marina.

Este proyecto forma parte de *ConCreTe* y está financiado por el programa *Future and Emerging Technologies* (FET), englobado en el *Seventh Framework Programme for Research of the European Commission* con número de subvención 611733.

Gracias a todos los anteriormente mencionados, ya que sin ellos este trabajo no habría sido el mismo.

Resumen

Generación de Lenguaje Natural a partir de Grafos Semánticos

Hoy en día la cantidad de información de la que el mundo dispone es inmensa y la gran mayoría está al alcance de un click gracias a las tecnologías de la información. Muchos de los recursos que existen en internet están escritos a mano por personas y para personas, pero este hecho tiene muchas limitaciones, como el idioma, el contenido, las expresiones en la comunicación o la disposición de la información en el texto. Todos estos factores influyen en el lector permitiendo entender mejor o peor los conceptos, relaciones e ideas que se expresan. Un ejemplo de un recurso muy utilizado a día de hoy es Wikipedia[1], que cuenta con más de cinco millones de artículos en inglés y más de un millón en otros doce idiomas entre los cuales se encuentran el castellano, el francés y el alemán.

Por otro lado, existen otros recursos que aportan información de otras formas más interesantes desde el punto de vista de la informática, como pueden ser ConceptNet[2] o WordNet[3]. Las ventajas que ofrecen este tipo de recursos son que no disponen de varios lenguajes, es decir el conocimiento está unificado en uno solo, no tienen estructura de texto y se puede automatizar más fácilmente la inserción de nueva información, lo que se traduce en un crecimiento más rápido del conocimiento. Este tipo de recursos son ideales para su uso en aplicaciones informáticas gracias a que no es necesario un proceso de extracción de información de la fuente. Sin embargo, este tipo de información no está pensada para la lectura por parte de un humano, ya que se enfrentaría a muchos datos de golpe y sin un orden lógico para la comprensión, además de carecer de la conjugación propia o traducción a un idioma concreto.

[1] `www.wikipedia.org`
[2] `www.conceptnet5.media.mit.edu`
[3] `www.wordnet.princeton.edu`

Este trabajo tiene como objetivo principal partir de un recurso de información no legible ni manejable por humanos e ideado para el uso por computadoras, y dar lugar a una interpretación de esta información que permita la lectura y comprensión en lenguaje natural por personas. Podemos verlo como un trabajo que posibilita y facilita el entendimiento Máquina-Hombre. Para ello se hace uso de un sistema de generación de lenguaje natural, inteligencia artificial y de la creatividad computacional. Además, este trabajo forma parte de un proyecto mayor, del que hablaremos en la sección 2.5, en el que se generan nuevos conceptos a partir de otros. El papel que desempeña esta aplicación permite describir los nuevos conceptos generados y poder entenderlos.

A la hora de abordar el problema de la generación de texto podemos encontrar varias formas de atacar la cuestión, y todas las soluciones se pueden considerar como válidas. Se implementarán sistemas de diferente complejidad y naturaleza, como generadores básicos de textos o generadores con planificación y otras soluciones comunes en este campo como el uso de plantillas y el estudio de las propiedades de los textos generados por los humanos. Por esta razón, en este trabajo se desarrollarán varios métodos y se valorarán según ciertos criterios como la claridad del texto, su organización, o si se ha hecho un buen uso de la gramática o la ortografía.

Como objetivos secundarios de este proyecto podemos remarcar la generación de un servicio web que permita que esté disponible la aplicación para su uso, y aporte valor tanto al mundo de la investigación como al del conocimiento. También se valora la semejanza a los generados por humanos.

Palabras clave

- Generación de Lenguaje Natural (GLN)
- Creatividad computacional
- Planificación de texto
- Realización superficial
- Grafos semánticos
- Tripletas

Abstract

Natural Language Generation based on Semantic Graphs

Nowadays the amount of available information is inmense and the huge majority of it is at the distance of a click thanks to the information technology. Lots of the existing resources on the internet are written by hand by people and for people, but this fact has lots of limitations, such as the language, the contents, the communication expressions or the order of the information in the text. All these factors influence the reader allowing them to understand in a better or worse way the concepts, relations and ideas that are expressed. An example of a widely used resource today is Wikipedia [4]. It counts with more than five million articles in English and more than a million in twelve more languages including Spanish, French and German.

On the other hand, other resources exists that provide information in more interesting ways from the computer science's point of view, like ConceptNet [5] or WordNet [6]. The advantages this kind of resources present are the unavailability for several languages, what means that the knowledge is unified in a single one, they have no text-based structure and the insertion of new information can be automatized more easily, what is translated to a quicker growth of the knowledge. This type of resources are ideal for their use in computing applications thanks to the fact that a process of information extraction form the source is not necesary. Nevertheless, this kind of information is not thought for human reading, as they will have to face lots of data at once and without a logical order for their comprehension. In addition, they lack the necesary conjugation and the translation to a specific language. Those languages are other than English which is the language in which all resources are written.

[4] `www.wikipedia.org`
[5] `www.conceptnet5.media.mit.edu`
[6] `www.wordnet.princeton.edu`

This project has as its main objective to begin from a non-legible or no-human manageable source of information prepared for its use by computers, and get an interpretation of this information that allows the reading and comprehension in natural language by people. We can see it as a work that makes real and easier the understanding between men and machines. To drive that, a natural language generation system is used along with, artificial intelligence and computational creativity. Furthermore, this application is part of a larger project in which new concepts are created starting from others. The role of this system let us describe the newly generated concepts and understand them.

When addressing the text generation problem we can find several paths to drive into the problem, and all answers can be considered valid. Systems with different complexity and nature will be implemented, such as basic text generators or generators with planning and other common solutions used in this field like the use of canned text and the study of the properties of the human-generated texts. For this reason, in this work several methods will be developed and they will be valued following certain criteria such as the clearness of the text, the text sequence, or if grammar and ortography has been used properly.

As secondary objectives of this project it is remarkable the generation of a web service that allows the application to be available for its use, and contribute to both the research word as well as the knowledge world. It will be rated positively the resemblance to the texts generated by humans.

Keywords

- Natural Language Generation (NLG)
- Computational creativity
- Text planning
- Surface realization
- Semantic graphs
- Triplets

Índice

1. Introducción **1**

1.1. Motivación 2

1.2. Objetivos 2

1.3. Plan de trabajo 4

1.4. Estructura del documento 4

1. Introduction **7**

1.1. Motivation 7

1.2. Objectives 8

1.3. Working plan 9

1.4. Document structure 10

2. Estado del arte **13**

2.1. Generación de Lenguaje Natural 14

2.2. Grafos semánticos 17

2.3. Tripletas 18

2.4. Creatividad computacional 20

2.5. ConCreTe 21

2.5.1. *ConCreTe Flows* 22
2.5.2. Contribución en la extracción de información 25
2.6. Recursos lingüísticos 27
2.6.1. WordNet 28
2.6.2. ConceptNet 5 30
2.6.3. DBpedia 32
2.6.4. Thesaurus Rex 33
2.6.5. TextStorm 34
2.6.6. Conclusiones de los recursos 36

3. Generación de texto a partir de información conceptual: *Textifier* 37

3.1. Entrada 38
3.2. Arquitectura 40
3.3. Implementación 44
3.4. Salida 45
3.5. Conclusiones del *Textifier* 46
3.6. Próximo sistema de generación de texto 47

4. Procesamiento de la entrada 49

4.1. Main entity extractor 50
4.2. Statistics 52
4.3. Relation simplifier 53
4.4. Relation unifier 55
4.5. Relation sequencer 57

5. Generación de texto **61**

5.1. Generando texto de forma básica 61

5.2. Estudio de textos generados por usuarios 64

5.2.1. Cuestionario corto 65

5.2.2. Cuestionario largo 66

5.2.3. Conclusiones del experimento 67

5.3. Generación elaborada de texto 68

5.4. Generación de texto con planificación 72

5.4.1. Planificador simple 72

5.4.2. Realizador de texto 74

5.5. Generación de texto simplificada 76

5.6. Comparación de los generadores 79

6. Conclusiones **83**

6. Conclusions **87**

7. Trabajo futuro **91**

Bibliografía **93**

A. Configuración de *ConCreTe Flows* **97**

B. Improving Information Extraction from Wikipedia Texts using Basic English **101**

C. Resultado de consulta a ConceptNet 5 **109**

D. Resultado de consulta a Thesaurus Rex **117**

Índice de figuras

2.1. Arquitectura de un sistema de generación de lenguaje natural 15

2.2. Grafo semántico 18

2.3. WordNet 29

2.4. ConceptNet 31

2.5. DBPedia 32

2.6. Thesaurus 34

3.1. Diferencias entre multigrafos 38

3.2. Grafo de los datos de entrada de Textifier construido 41

3.3. Pipeline del Textifier 45

3.4. Arquitectura del sistema 48

4.1. Grafo semántico sobre "hamster" extraído por TextStorm . . . 55

4.2. Grafo semántico sobre "hamster" extraído por TextStorm con las relaciones simplificadas 56

4.3. Grafo semántico sobre "hamster" extraído por TextStorm con las relaciones simplificadas y unificadas 57

5.1. Diagrama de instrucciones 73

5.2. Hiperonimia 77

Índice de Tablas

2.1. Principales relaciones en ConceptNet 5 expresadas con URI . 31

4.1. Tabla de dispersión y calidad de los datos en varios recursos y conceptos 53

4.2. Tabla de traducción de relaciones 54

4.3. Tripletas sobre "hamster" extraídas por TextStorm 56

4.4. Tripletas sobre "hamster" extraídas por TextStorm con las relaciones simplificadas 57

4.5. Tripletas unificadas con el formato interno del sistema 58

5.1. Tripletas disponibles para la encuesta corta 65

5.2. Tripletas disponibles para la encuesta larga 66

5.3. Instrucciones generadas para la generación de texto con planificación simple 74

5.4. Valoraciones de los generadores 82

Capítulo 1

Introducción

Saber comunicarse es un proceso de aprendizaje lento que las personas comenzamos desde que nacemos. Cada persona se comunica de diferente manera al resto y esta evoluciona y cambia según la etapa de la vida en la que se encuentre. Esto sugiere que no existe una meta a la que llegar en este aprendizaje y que toda comunicación es válida. La diferencia está en la efectividad del envío del mensaje.

En el área de la informática existe un campo de estudio llamado *"Generación de Lenguaje Natural"* (GLN) (más conocida en inglés como *"Natural Language Generation"* (NLG)) que se ocupa de simular el proceso de comunicación humana. Se pueden abordar diferentes problemas gracias a este área, como es la generación de descripciones o generación de noticias, pero también se adentra en el arte con generadores de cuentos, adivinanzas o poesías.

Hoy en día, intentamos generar lenguaje natural de muchas formas diferentes y para diferentes tipos de receptores. Las investigaciones que se realizan tratan sobre aspectos como la creatividad o la originalidad del texto generado.

Como podemos ver, la gran mayoría del trabajo se escribe y se desarrolla para un idioma: el inglés. Por esta razón, en muchas ocasiones, los ejemplos de datos de entrada, de salida, código y palabras clave que veremos vendrán en dicho idioma.

1.1. Motivación

Hoy en día la cantidad de información que existe en internet es inmensa. Parte de esa información está escrita por las personas directamente y puede ser leida y comprendida por cualquier lector de la red. Sin embargo, existe también mucha información que está almacenada de forma que resulta sencillo de utilizar por un programa. Podemos llamarlo recurso. Estos recursos son representaciones semánticas de la información y tienen diferentes formatos y diferentes tipos de información, lo que quiere decir que se tienen que estudiar y explotar por separado o realizar alguna transformación para unificar el conocimiento. Este programa puede utilizar dicha información para consultarla, pero también podría contribuir a mejorar o simplemente cambiar esa información. Toda esta información permanece inaccesible para las personas, por lo que resulta interesante poder ser capaz de transformar un recurso en texto comprensible por humanos.

La calidad de un texto generado automáticamente se puede medir por la facilidad de comprensión del mismo. Es por ello que en este trabajo vamos a procurar realizar textos descriptivos sencillos de entender y que intenten tener la naturalidad esperada de una persona.

Por otro lado, este trabajo quiere contribuir a un proyecto de investigación llamado *ConCreTe*, que se explicará más adelante en la sección 2.5 [1] destinada a ello. Esto nos llevará a decisiones en el sistema para poder adaptarlo a las necesidades del proyecto. En este proyecto se quiere conseguir crear conceptos nuevos a partir de conceptos ya existentes, y el resultado comparte información de ambos. Nuestro objetivo es recibir el nuevo concepto que se ha creado y realizar una descripción para poder explicarlo. No debería haber diferencia en la generación de las descripciones al explicar conceptos nuevos o conceptos conocidos. Así que realizaremos de la misma forma la generación del texto. Tomar información sencilla que represente conocimiento conocido para construir y probar el sistema nos facilitará el desarrollo del sistema, y una vez terminado será cuando realizaremos las pruebas con los conceptos nuevos (generados desde la plataforma del proyecto).

1.2. Objetivos

Este trabajo se encarga de poder hacer legible la información de los recursos de internet que están preparados para su uso por computadoras. El

[1] `http://conceptcreationtechnology.eu/`

primer paso para conseguir este objetivo es localizar dichos recursos que resulten más óptimos y útiles, ya que algunos de los recursos de información pueden no poder dar lugar a un texto descriptivo que hable de la información de una forma coherente. Estos recursos no tienen una estructura ni un orden lógico para la compresión humana, pero al construir un texto que explique dicha información es necesario utilizar una estructura, ya que la lectura de un texto siempre es secuencial y tiene continuidad desde el principio hasta su fin. Por tanto, es necesario seleccionar bien el orden en el que se colocan las ideas y cómo se explican los conceptos para facilitar o posibilitar la tarea del lector.

Otros aspectos que también se valorarán positivamente son la naturalidad, la originalidad y otros aspectos que mejoren la calidad final del texto.

La aplicación que se va a desarrollar en este trabajo se va a hacer uso en el proyecto llamado *ConCreTe*, para generar una de sus salidas. Esta salida es un texto que pretende explicar un concepto nuevo creado por el sistema del proyecto. Por esta razón es necesario que la aplicación que se desarrolle tenga cabida en dicho proyecto y se amolde a sus necesidades. Todo ello sin perder de vista la generación de lenguaje natural a partir de otros recursos de la web.

Los sistemas informáticos tienden a evolucionar y cambiar según sus objetivos y necesidades. Debido a esto, procuraremos tener una arquitectura de la aplicación lo más preparada posible para cambios y un código extensible que podamos reutilizar en caso de necesitarlo.

Ahora enumeraremos los objetivos principales de este trabajo:

- Estudiar los recursos existentes en internet para conseguir localizar fuentes de conocimiento de donde poder extraer grafos de conocimiento fiables y fácilmente manejables.
- Realizar los cambios necesarios en los datos de entrada para transformar la información a un formato interno del sistema, siendo así independiente del recurso, y a su vez facilitar la generación del texto.
- Crear un generador de texto para los diferentes datos de entrada posibles al sistema de generación de lenguaje natural. Debe describir los conceptos para hacer accesible estos a las personas.
- Incluir el sistema de generación en la tubería del proyecto *ConCreTe* al que pertenece y contribuye.

1.3. Plan de trabajo

En esta sección vamos a hablar del plan de trabajo, que es la organización en el tiempo para el desarrollo e implementación de la aplicación y estudios necesarios que complementan este trabajo.

El primer paso es estudiar el estado del arte (capítulo 2) o estado de la cuestión, donde leeremos y nos informaremos de todos los avances e investigaciones publicadas hasta ahora en los campos de la informática que vayamos a utilizar. Tendremos que tener en cuenta cuales son las técnicas que mejor funcionan para el problema que queremos resolver, normalmente los estudios más recientes son los que muestran mejores comportamientos.

Una vez hemos estudiado y decidido lo que vamos a utilizar para llevar a cabo el trabajo podemos comenzar el segundo paso. En este estudiaremos los recursos de conocimiento disponibles (sección 2.6) en internet, y veremos las características de cada uno, teniendo en cuenta las ventajas y desventajas que supondría utilizar cada uno de ellos. Terminaremos este estudio decidiendo cuáles son los que vamos a utilizar para nuestro trabajo.

En tercer lugar desarrollaremos una serie de funcionalidades de preprocesamiento (capítulo 4) de los datos de entrada que tenemos para adecuarlos al formato interno del sistema, lo que nos permitirá abstraernos del recurso concreto que estemos utilizando para generar el texto. En este mismo paso, crearemos otras utilidades de preprocesamiento con las que prepararemos los datos para poder generar el texto de una forma más sencilla.

Por último pasaremos al cuarto paso. En este crearemos un generador de texto. Repetiremos este paso hasta tener un generador de texto que cubra todas nuestras expectativas y las del proyecto. Una vez tengamos una versión completamente funcional pasaremos a incluir la aplicación en el proyecto.

1.4. Estructura del documento

Este trabajo está organizado de forma que primero se explica el estado del arte, en el capítulo 2. En este capítulo, podemos encontrar un estudio de los avances publicados hasta ahora en los campos de la informática que vamos a utilizar, lo que va a ayudar a poder enfocar mejor el trabajo y a saber cómo y por dónde empezar. También encontraremos la descripción del proyecto, cómo configurarlo para utilizarlo y contribuir en él y las contribuciones que nos pueden influir. Al final del capítulo, encontraremos un estudio

de recursos útiles para tenerlos en cuenta como entrada o utilizarlos como ayuda adicional en otro proceso del sistema.

En el capítulo 3 se explica el desarrollo de una aplicación para la generación de lenguaje natural para el proyecto *ConCreTe*, mencionado anteriormente. Esta herramienta pertenece a una fase intermedia del proyecto y tras decisiones entre los colaboradores en la investigación que se lleva a cabo se decide modificar drásticamente la información proporcionada al generador de texto. Este cambio en la entrada de nuestro sistema nos lleva a replantear la solución, por lo que veremos en el capítulo 4 las modificaciones y utilidades que podemos aplicar a los datos de entrada para facilitar y mejorar nuestra tarea, la generación. En el capítulo 5, explicaremos las diferentes propuestas para abordar el nuevo problema, veremos su funcionalidad y resultados.

Por último, en los capítulos 6 y 7 encontraremos las conclusiones y el trabajo futuro de este proyecto.

Chapter 1

Introduction

Communicate properly is a large learnig process that people begin since we were born. Each person express themself in a different way, it evolves and changes depending on the life's phase in which they are. This suggest there is no goal to reach in this learning task and every way of communication is valid. The difference between them is in the effectiveness of sending the message to the receiver.

In the computer science area there is a field called *"Natural Language Generation"* (NLG) that its purpose is to simulate the human communication process. Thanks to this area, problems like description generation or news generation can be adressed. But we also can create art with tail, riddle or poetry generators.

Today's tendency is to grenerate natural language in many different ways and to different kind of receivers. The reserch that are being performed deal with the creativity or the originality of the generated texts.

As we can see, the vast majority of the work is written and developed for a concrete language: English. For this reason, in several occasions, the input and output example data, the code and the key words we are going to see will be in that language.

1.1. Motivation

Nowadays, the quantity of the information that exists on internet is immense. A part of that information is written by people directly and it can

be readed and comprehended by any user on the net. Nevertheless, a lot of information is stored in a way that only computers can easily use. We can call them resources. Those resources are semantic representations of the information and they have different formats and different types of information, this means they have to be studied and used separately unless a unification transformation of the knowldge is executed before. A program can use this available information for consulting, but also it can contibute to improve or simply to transform that information. Every piece of this information remains inaccessible to ordinary people, that is why it is interesting to be capable of transforming a resource in a comprehensible text for humans.

The quality of an automatically generated text can be measured by its ease of comprehension. That is why in this work we will try to perform simple descriptive texts to understand, trying to have the expected naturalness of a person.

On the other hand, this work wants to contribute to a research project called *ConCreTe*, which will be explained lately in section 2.5 [1] which is intended to do so. This will address us to some decisions in the system so that we can adapt it to the necessities of the ptoject. This project want to create new concepts from others that already exists, the new concept is made by a mix of information from the other ones. Our objective is to receive the new generated concept and create a description to explain it. It should not be a difference between the generation when describing new or known concepts. For this reason we will use the same mechanism to generate both of them. Using simple information representing known information to build and test the system will ease its develompent. Once it is finished, we will do tests with the new concepts (generated by the project's platform).

1.2. Objectives

This work is responsible for making legible the information of the resources on internet that are prepared to be used by computers. The first step to achieve this objective is to locate those resources that are optimal and useful, because some of the information resources can not be able to be transformed to a descriptive text which express all the information in a coherent way. The reseources we are talking about has no structure or logic order for human comprehension, but when building a text that explains such information a structure is needed, because reading is always a sequential task and it has continuity from the beginning to its end. Therefore, to use a good

[1] `http://conceptcreationtechnology.eu/`

order for the ideas in the text is needed, but also how to explain the concepts in it make possible or facilitate the reading.

Other aspects that will also valued positively are naturalness, originality and aspects that improve the quality of the final text.

The application is going to be developed during this work is going to be used in the project named *ConCreTe* to generate one of its outputs. This output is a text that aim to explain a new concept created by the system of the project. For this reason, the developed application needs to fit in the project and mold itself to its necesities. With this in mind, we have to create the natural language generator for the other resources from the web.

The computing systems tend to evolve and change depending on their objectives and needs. Due to that, we will attempt to have an application architecture prepared to possible changes and a extensible code we can reuse if needed.

Now we are going to enumerate the main objectives of this work:

- Study the existing resources on internet to locate the sources of knowledge where we can get reliable knowledge graphs which are easy to use.
- Make the necessary changes in the input data to transform the information into an internal system format, thus it is independent of the resource, and at the same time facilitating the text generation task.
- Create a text generator for every possible input data to our natural language generation system. The concepts must be described to allow people understand them.
- Include the generation application in the pipeline of the project, *ConCreTe*, it belongs and contributes to.

1.3. Working plan

In this section we are going to talk about the working plan, it is the time organisation for developing and implementing the application and needed studies that complements this work.

The first step is to study the state of the art (chapter 2), where we will read about all advances and published research until now in the computing

fields we are going to use. We will have to consider the techniques that work better to solve the problems we have to face, usually the most recent studies are the ones that show better behaviours.

Once we have studied and decided roughly what we are going to use to accomplish the tasks, we can begin the next step. In this one we will study the knowledge resources available (section 2.6) on internet and we will evaluate the characteristics of each one comparing the advantages and disadvantages that they will provide. We will finish this study by deciding which are the ones we are going to use for our system.

In third place we will develop a group of preprocessing functionalities (chapter 4) for the input data we have, to adapt them to the internal system. This will allow us to forget about the resource we are using to generate the text. In this step, we will also create other preprocessing functionalities to prepare the data to the text generation task and make this work easier.

Lastly, we have the fourth step. In this step we will create a text generator. We will repeat this process until we have a text generator that fulfill every of our expectations. When we have a completely functional version we will include the application to the project.

1.4. Document structure

This work is organised so that the first thing is explpained is the state of the art, in chapter 2. In this chapter, we can find a study about the last published advances until now in the computing fields we are going to use. This will help us to approach better the work and to know how and where to start. We will also find the description of the project, how to configurate it, how to contribute to it and the contributions that may influence us. At the end of this chapter we will find a study of useful resources that could be used as input or as helping tools in the system.

In chapter 3 it is explained the development of a natural language generation application design for the *ConCreTe* project we mentioned before. This tool belongs to an intermediate phase of the project, but after taking some decisions between the partners in the current research, we decided to change drastically the information provided to the text generator. In response to this change in the input of our system, we have to reconsider the solution. In chapter number 4 we will see the modifications and utilities we can apply to the input data to ease and improve our task, the generation. In chapter 5, we will explain the different approaches to aim the new problem, we will also talk about their functionality and results.

Lastly, in chapters 6 and 7 we will find the conclusions and the future work of this project.

Capítulo 2

Estado del arte

La informática es una ciencia relativamente joven y que crece muy rápido. Por esta razón, para utilizar las últimas técnicas, más novedosas y mejores, se deben estudiar los avances más recientes relacionados con nuestros objetivos. Dado que el camino a la meta de este trabajo está directamente relacionada con la generación de textos en lenguaje natural, veremos de qué partes consta un sistema de este tipo y cómo funciona en la sección 2.1. También tendremos que investigar sobre la representación de la información, para poder crear buenas estructuras que representen los datos con los que vamos a trabajar. Esto lo veremos en las secciones 2.2 y 2.3. Existen propiedades de los textos que proporcionan características propias de un texto generado por humanos, el campo de la informática que aborda esto es la creatividad computacional, hablaremos de ella en la sección 2.4. También tenemos que tener en cuenta el conocimiento existente del que podemos hacer uso, para ello veremos una serie de recursos y sus características en la sección 2.6. Estos serán los primeros temas sobre los que investiguemos. Tras el estudio del estado de la cuestión, podremos utilizar lo aprendido para empezar a desempeñar las primeras tareas y decisiones de este proyecto.

No podemos olvidar que este trabajo pertenece a un proyecto de magnitud más grande. Este es uno de los proyectos de investigación más actuales sobre los campos que acabamos de mencionar, además de otros muchos. Por esta razón, debemos amoldarnos a la entrada que este nos proporcione y adecuarnos también a la salida que se espera. Estudiaremos y explicaremos las características de este proyecto y las contribuciones que nos pueden influir en el capítulo 2.5.

2.1. Generación de Lenguaje Natural

La Generación de Lenguaje Natural es un campo de la informática con mucha actividad en investigación. Ésta aborda el problema de generar lenguaje humano. Los factores que influyen en este problema y durante el proceso que se sigue computacionalmente son la información disponible (base de conocimiento), las herramientas lingüísticas, el objetivo comunicativo y el receptor (Reiter et al., 2000). Típicamente el proceso que se sigue se basa en dos fases, la planificación del texto y la generación o realización. En esta última parte se pueden diferenciar una parte léxica y otra gramatical (Reiter et al., 2000).

Para entender de dónde surge esta necesidad de la informática vamos a mencionar algunos de los acontecimientos más memorables en la historia, en orden cronológico. Tal como indica el famoso libro *"Building Natural Language Generation Systems"* (Reiter et al., 2000), este joven campo de la informática tiene su comienzo en torno a las décadas de 1950 y 1960 en el contexto de proyectos de traducción. En la década de los sesenta se pudieron ver los primeros intentos utilizando gramáticas de lenguaje natural como una forma de generar frases bien formadas de forma aleatoria. Más tarde, en los setenta aparecieron las primeras tesis doctorales, como la de (Goldman, 1975) que se centra en la elección de las palabras para describir contenido conceptual, y la de (Davey, 1979) que busca estructuras de texto y expresiones de referencia de forma apropiada para descripciones de juegos del estilo *tic-tac-toe*. Más adelante, a principios de la siguiente década, se notó un gran aumento en el número de estudios. Podemos remarcar el trabajo de (Appelt, 1985) y de (McKeown, 1985). Fue entonces cuando se decidió cambiar el tipo de aplicaciones para abordar los problemas de este campo. Antes se construían sistemas grandes que se encargaban de todas las fases y aspectos de la generación. A partir de este momento, se generalizó la implementación de sistemas más pequeños que tratan problemas concretos. Al crecer la comunidad surgieron congresos específicos sobre la generación de lenguaje natural, el primero de los cuales fue el "*International Workshop on Natural Language Generation*" en 1983.

En el libro *"Building Natural Language Generation Systems"* (Reiter et al., 2000), que ya hemos mencionado con anterioridad, se propone la construcción de un sistema con tres módulos. A estos módulos se les llama planificador de documentos (*document planner*), microplanificador (*microplanner*) y realizador superficial (*surface realiser*), que se ejecutan de forma secuencial y en el orden en el que los hemos nombrado. La entrada al primer módulo es un conjunto de información que representa un objetivo de comunicación

y la salida el texto (*surface text*). En la figura 2.1 se puede ver un esquema de un sistema de generación natural junto con su entrada y salida.

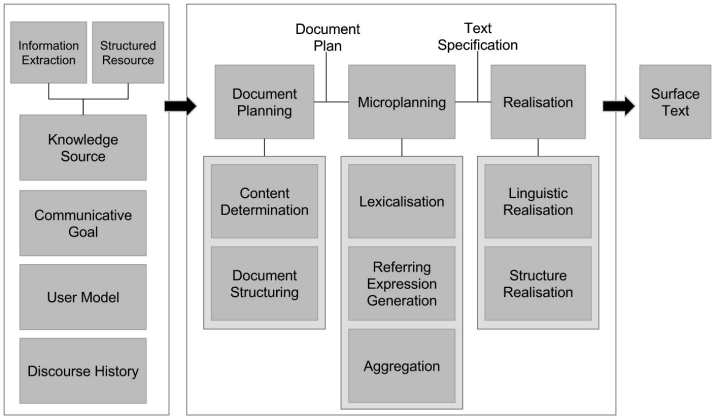


Figura 2.1: Arquitectura de un sistema de generación de lenguaje natural

La entrada que se propone a un sistema de generación de este tipo es una tupla de cuatro componentes: La fuente de conocimiento (*knowledge source*), el objetivo de comunicación (*communicative goal*), un modelo de usuario (*user model*) y una historia de discurso (*discourse history*). Esta tupla se representa como $<k,\ c,\ u,\ d>$

El planificador de documentos se encargará de la parte de determinación de contenido (*content determination*) y de darle una estructura del documento (*document structure*). A continuación veremos con un poco más de detalle estas tareas:

- En la determinación de contenido nos ocuparemos de decidir qué información se incluirá y cuál no en el texto final, es decir, las ideas que queremos transmitir.
- La tarea de estructuración del documento es la que se encarga de decidir cómo se agrupan las ideas, se decide separar el texto en trozos en los cuales cada uno tiene un conjunto de ideas coherentemente elegido.

La salida del módulo de planificación de documentos es a su vez la entrada al microplanificador. Se le llama plan de documento (*document plan*).

El microplanificador tiene como tareas sobre el contenido la lexicalización (*lexicalisation*) y la generación de expresiones de referencia (*referring expression generation*). La única tarea de estructura de la que se encarga este módulo es la agregación (*aggregation*). Ahora vamos a explicar los objetivos de cada una de ellas:

- La lexicalización es el proceso que elige las palabras que se van a utilizar para expresar el contenido que se ha decidido incluir en el texto. También decide la estructura sintáctica que se va a utilizar.
- La tarea de generación de expresiones de referencia es la encargada de decidir cómo se va a hacer referencia a las entidades o conceptos que aparecen en el texto.
- Para poder pasar de la estructura decidida por el planificador a una estructura sintáctica de la lengua existe la tarea de agregación. Esta decidirá cómo se organiza el texto en términos de frases, párrafos y el orden en el que aparece la información.

Una vez finalizado el procesamiento de los datos en el módulo del microplanificador la estructura de datos que obtenemos se llama especificación del texto (*text specification*) y será la entrada al último módulo de la generación de lenguaje natural.

Las tareas de realización lingüística (*linguistic realisation*) y realización estructural (*structure realisation*) son las que se encuentran en el módulo del realizador superficial. Ahora profundizaremos un poco en ambas:

- En la realización lingüística pasamos de tener una representación de la información interna del sistema a una representación del texto final, que corresponde a la salida.
- Los signos de puntuación que determinan frases, párrafos apartados y similares se generan gracias a la tarea de realización estructural.

Cabe destacar que a la hora de especificar las frases, que son los nodos hoja en la especificación del texto, podemos optar por varias formas, dependiendo de la elección serán necesarios ciertos pasos para la construcción de la frase final. Las cadenas ortográficas es una de las opciones, esta contiene la frase con las mayúsculas y signos de puntuación necesarios, y no necesita procesamiento posterior. Otra de las posibles elecciones que tenemos es el uso de plantillas, más conocidas como *canned text*, que utilizan frases predeterminadas para rellenar el texto. Este tipo de frases no contiene las reglas ortográficas que requeriría el texto final, por lo que debe pasar por un proceso donde se añadan los signos de puntuación necesarios y se haga un correcto uso de las mayúsculas. La última alternativa de la que vamos a hablar es la estructura sintáctica abstracta. Este tipo de representación contiene el lexema del concepto y un conjunto de características que debe cumplir para poderse generar el texto final, esta solución es la que más procesamiento

adicional requiere. Existen otros tipos de representación de los que no hemos hablado y también es válido el uso de cualquier mezcla o combinación de representación.

2.2. Grafos semánticos

Un grafo semántico (*semantic graph*) o red semántica (*semantic neworks* (Sowa, 1987)) permite representar relaciones semánticas entre conceptos y se utiliza para representar grafos de conocimiento (*knowledge graphs*), lo que resulta muy útil en el campo de la generación de lenguaje natural. Podemos encontrar diferentes formas de representar este tipo de estructuras, representando los conceptos como nodos y las relaciones también como nodos o con enlaces. Las relaciones pueden ser tanto unidireccionales como bidireccionales, según se especifique, pero lo normal es que sean dirigidas ya que en semántica no suele ocurrir que exista la relación en ambos sentidos. A continuación, en la figura 2.2 podemos ver un ejemplo de grafo semántico. En el grafo semántico de ejemplo podemos ver que representa información acerca de animales. Este grafo es dirigido. Gracias a este sencillo grafo podemos conocer información como que el *pez* y la *ballena viven en* el *agua*, que el *gato* y el *oso* además de compartir la relación de que *tienen pelo*, ambos son *mamíferos*. También podemos categorizar al *gato*, el *oso*, el *pez* y la *ballena* como *animales*. Con este grafo podríamos construir un diagrama de Venn para representar esta información y visualizarla con más claridad para una persona, y así reconocer las partes comunes entre cada nodo del grafo. Por supuesto, esta información de ejemplo no es totalmente representativa. Existen grafos semánticos que en cada nodo o relación tienen más información que una palabra o concepto: pueden contener frases parciales o totales que expresen ideas. Un pequeño extracto de un grafo con estas características podría tener un nodo con "los coches de lujo" como información, que se une a otro nodo que contiene "en zonas rurales" por medio de la relación "son raros de ver".

Una forma muy común para crear un grafo y rellenarlo de conocimiento es la que siguen en (Leskovec et al., 2004). Se comienza seleccionando el texto (escrito en lenguaje natural) del que se desea extraer la información. También se pueden crear grafos a partir de otras fuentes estructuradas pero en este caso el trabajo se resumiría en seleccionar y copiar el contenido. Luego se recorren las oraciones una a una desde el principio hasta el final extrayendo tripletas que representan la información del texto y se ejecutan las tareas de postprocesado llamadas *resolución de pronombres en oraciones cruzadas* (*cross-sentence pronoun resolution*), *resolución de co-*

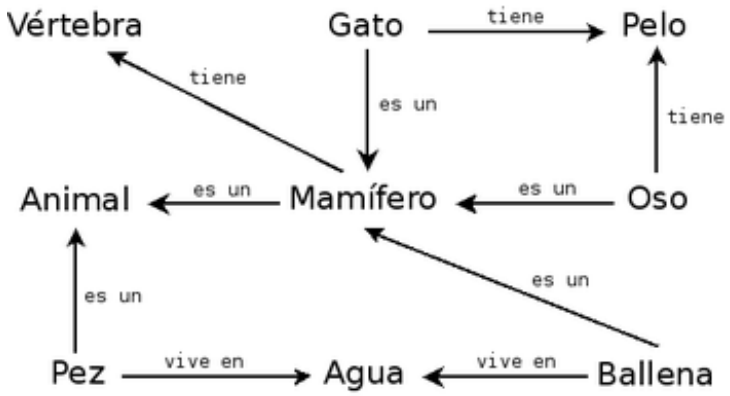


Figura 2.2: Ejemplo de grafo semántico

referencias (*co-reference resolution*) y normalización semántica (*semantic normalization*).

Una muy buena práctica es restringir las relaciones que se pueden representar, con lo que obtenemos un grafo más conexo y podemos relacionar mejor la información. Principalmente se estudió en el proyecto *Knowledge Graphs* en los que participan dos universidades de los Países Bajos (Groningen y Twente), podemos consultarlo en (van de Riet y Meersman, 1992). En la creación de un grafo de este tipo existen varias fases: análisis del texto (text analysis), identificación de conceptos (*concept identification*), integración de conceptos (*concept integration*) e integración de enlaces (*link integration*).

Un ejemplo de uso de este tipo de estructuras podemos verlo en un artículo sobre extracción de frases de resumen ((Leskovec et al., 2005)). En este trabajo se utilizan tripletas con un sujeto, un objeto y un predicado para crear el grafo.

2.3. Tripletas

Una tripleta es un tipo especial de tupla, en concreto de tres datos que son comúnmente cadenas de texto, aunque se puede encontrar cualquier tipo de información. Como hemos visto en la sección 2.2, se utilizan para representar conocimiento y formar estructuras con información acerca de un concepto. Normalmente las tripletas tienen un único sentido, esto es, pueden leerse solo de izquierda a derecha. Si se lee de derecha a izquierda se trataría de otra tripleta con un significado diferente (no deseado). La estructura y orden de la representación puede variar según la implementación. Para referirnos a ellas de una forma concreta se muestra a continuación un modelo de

representación y damos nombre a las tres partes para poder diferenciarlas fácilmente.

(Concepto1 - Concepto2 - Concepto3)

- Concepto1: Este es el "concepto sujeto" que es el concepto principal de la tripleta y funciona como sujeto de la relación que se representa en este modelo. También puede llamarse parte izquierda de la tripleta, ya que ese es el sitio que le corresponde horizontalmente.
- Concepto2: Es la "relación", esta une las partes A y B de la tripleta. Desempeña la función de verbo entre ambos, se representa como un verbo en infinitivo.
- Concepto3: Esta es la parte derecha de la tripleta, que llamaremos "concepto objeto". Este concepto es sobre el que recae la acción del verbo.

Muchos de los recursos de conocimiento de la web proporcionan la información en forma de tripletas o permiten fácilmente la extracción de estas (algunos de los recursos existentes los veremos más adelante en el apartado 2.6). Un sencillo ejemplo de tripleta es el siguiente:

(hamster - Is_A - rodent)

Si queremos representar el grafo semántico 2.2 que hemos visto anteriormente en 2.2, podemos hacerlo con tripletas como se muestra a continuación.

(Ballena - Es_Un - Mamífero)
(Ballena - Vive_En - Agua)
(Pez - Es_Un - Animal)
(Pez - Vive_En - Agua)
(Mamífero - Es_Un - Animal)
(Mamífero - Tiene - Vértebra)
(Oso - Es_Un - Mamífero)
(Oso - Tiene - Pelo)
(Gato - Es_Un - Mamífero)
(Gato - Tiene - Pelo)

Uno de los mucho trabajos interesantes que utilizan tripletas para la generación natural de texto es el descrito en (Krishnamoorthy et al., 2013). Como podemos observar, este artículo está publicado hace pocos años, lo que implica que es una forma de trabajo que hoy en día se utiliza y resulta común en la comunidad. En este trabajo se desarrolla un estudio de investigación que extrae los objetos de imágenes obtenidas de un vídeo con una herramienta (Felzenszwalb et al., 2008). Luego, gracias a otra herramienta

desarrollada por (Laptev et al., 2008), extraen la acción del vídeo. Con estos datos crean tripletas y generan (mediante herramientas lingüísticas) con el verbo potenciales palabras que expandan su significado para mejorar la descripción, por ejemplo, de *move* podríamos obtener verbos como *walk*, *fall* o *follow*. Finalmente pasa los datos por una fase de planificación del contenido y realización superficial donde se hace uso de plantillas para generar el texto final. Realizaron una evaluación de varios aspectos, tanto de la habilidad del sistema para identificar el contenido como para la precisión de la información. Las evaluaciones de la generación de lenguaje natural son tanto necesarias como útiles, pero en muchos casos son costosas en tiempo y en recursos. En el experimento del que hablamos las evaluaciones mezclan evaluación con personas y automatizada.

2.4. Creatividad computacional

La creatividad computacional es un subcampo de la inteligencia artificial que tiene como meta simular la creatividad humana por medio de la informática, como se explica y detalla en (Colton et al., 2012). Ésta es una difícil tarea, ya que no podemos definir claramente qué es o qué no es la creatividad y cómo funciona, y tampoco sabemos muy bien cómo medirla o evaluarla de una forma objetiva. Existen algunos avances para valorar de una forma razonada ciertos tipos de creatividad, por ejemplo, en (Jordanous, 2012) se propone un método de evaluación para sistemas creativos. Este sistema se basa en tres pasos: la determinación de qué es creativo para el sistema a evaluar, derivación y representación de test basados en estas afirmaciones. Este sistema se ha probado para evaluar la creatividad de programas de improvisación musical, un aspecto muy concreto del campo. La psicóloga (Amabile, 1983) dijo *"If a machine can do only what it was programmed to do, how can its behavior ever be called creative?"*, lo que se podría traducir como *"Si una máquina solo puede hacer aquello para lo que fue programada, cómo podría su comportamiento llamarse creativo"*. Esta idea es acertada, ya que hasta ahora un programa solo actuará con el comportamiento esperado y podrá tomar decisiones para las que fue diseñado, pero no puede tener iniciativa propia, no puede evolucionar.

Se han realizado varias investigaciones por parte de (Fauconnier y Turner, 1998, 2008) sobre creatividad computacional, concretamente sobre fusión conceptual (*conceptual blending*). Para la realización de un sistema de este tipo, se propone una red de trabajo con cuatro espacios, dos entradas, un espacio predefinido de conocimiento general que permite entender la mezcla final de los dos conceptos y por último la mezcla, que puede haber solapado o sobre escrito información de cualquiera de los datos. La fusión conceptual

permite crear un nuevo concepto por medio de otros ya conocidos, y como resultado de esta creatividad se obtienen propiedades que no se tenían antes juntas en un mismo concepto.

El área de la lengua permite muchos tipos de creatividad como al generar historias (Méndez et al., 2016b) como para la interacción de los personajes de las mismas (Méndez et al., 2016a), o para construir figuras retóricas (Galván et al., 2016b). También se utiliza la creatividad para generar y/o reconocer sarcasmos e ironías (McDonald, 1999), adivinanzas (Galván et al., 2016a), chistes (Dybala et al., 2010), conversaciones (Cassell et al., 2000) e incluso musicales y contenido emocional (Gervás et al., 2016), como Beyond the Fence - The Musical[1].

Un reciente artículo (Cook y Colton, 2014), merece especial atención. En esta publicación se habla de un sistema llamado "ANGELINA-5", el cual se dice que es capaz de competir contra los humanos en términos de creatividad. Este sistema es capaz de diseñar y crear videojuegos simples en tres dimensiones de forma autónoma.

Existen muchos otros tipos de creatividad computacional como la creatividad visual , artística o la creatividad aplicada a solucionar problemas.

A pesar de todos estos avances que hemos mencionado y todas las características que se cubren de la creatividad humana, se dice que aún no está a la altura de suplantar las grandes obras artísticas producidas por personas. Aunque una cosa está clara, con cada avance en este campo la creatividad computacional está un paso más cerca de igualarlo y resulta más inteligente y menos artificial.

2.5. ConCreTe

El presente trabajo de fin de máster es parte de un proyecto europeo llamado *ConCreTe* [2] que alude a *Concept Creation Technology* (financiado por el programa *Future and Emerging Technologies* (FET), englobado en el *Seventh Framework Programme for Research of the European Commission* con número de subvención 611733). El objetivo principal de este proyecto es el estudio de la creatividad en personas y máquinas, y pretende simular la creatividad humana con ayuda de la informática donde el sistema será preferiblemente autónomo e interactivo. Las universidades que participan en

[1] `http://nil.fdi.ucm.es/?q=node/613`
[2] `http://conceptcreationtechnology.eu`

este proyecto son la Universidad Complutense de Madrid (España), Universidad de Coimbra (Portugal), Universidad de Helsinki (Finlandia), Instituto Jožef Stefan (Liubliana, Eslovenia), Universidad de Twente (Países bajos) y la universidad de Queen Mary de Londres (Reino Unido). Esta última es la que se encarga del papel de coordinador del proyecto. También colabora en el proyecto una empresa para dar salida a los resultados del proyecto, aunque también aportan ayuda en la investigación. Esta empresa es Chatterbox Labs Ltd (Londres, Reino Unido). Para la creación, modificación, configuración y ejecución de este sistema disponemos del llamado *ConCreTe Flows* donde están disponibles los módulos que se han añadido al sistema por parte de cada una de las universidades colaboradoras de este proyecto. En ConCreTe Flows se puede configurar el entorno de trabajo para utilizar como deseemos los módulos y conectarlos de diferentes formas para probar los diferentes resultados. Esta plataforma la explicaremos con más detalle en el apartado 2.5.1. Existen varias tareas que se pueden abordar con los módulos del sistema, como generación de poesía, generación de letras de canciones, generación de comparaciones y adivinanzas, generación resúmenes, generación de imágenes, generación de texto o extracción de información.

El principal objetivo de la Universidad Complutense de Madrid es el de generar texto en lenguaje natural, para poder explicar los nuevos conceptos que se crean y todas sus propiedades. Ésta es una de las dos salidas que tiene el sistema del proyecto. La otra es mediante composición de imágenes sobre los conceptos anteriores a la mezcla. La entrada de datos que se proporciona para la generación de la salida de texto es un grafo semántico con la información a describir. Esta estructura de datos ha ido evolucionando con el desarrollo del proyecto. A medida que se avanzaba en la investigación de la generación de los nuevos conceptos se planteaba una mejor representación de la información. Este hecho repercute directamente en la generación del texto de salida, ya que al cambiar la naturaleza y la complejidad de los datos es necesario abordar el problema de una forma diferente.

También contribuimos en otras tareas del proyecto, como la que se encarga de generación de resúmenes a partir de los grafos semánticos y la tarea de extracción de información para mejorar la calidad y cantidad de la información de la que se dispone. Esto último afecta directamente al texto final que se obtiene como salida del sistema. Entraremos a explicar esta parte de la contribución al proyecto en el apartado 2.5.2.

2.5.1. *ConCreTe Flows*

Como ya hemos mencionado, *ConCreTe Flows* es una herramienta que permite utilizar los recursos existentes proporcionados por los colaboradores

para el proyecto europeo. Esta, permite disponer del sistema completo del proyecto, probar sus componentes, ver los formatos de entrada y de salida de cada uno de los módulos de forma independiente. Además, de servir de herramienta interna también permite mostrar el flujo y los resultados a los encargados de evaluar los avances conseguidos. Con ella, todos los colaboradores e investigadores del proyecto pueden tener una copia local de todo el proyecto y su funcionalidad .Esto sirve para utilizar los módulos disponibles de forma que se pueden realizar pruebas sobre las aplicaciones que se están desarrollando con el contenido actual del proyecto y asegurarse el buen funcionamiento con los datos finales.

Algunos de los módulos que están actualmente disponibles en esta plataforma permiten la extracción de información a partir de una página web con información no estructurada de un concepto, como puede ser un artículo de una enciclopedia. Existe otro módulo que dados dos conceptos en una representación con tripletas o de grafo semántico, devuelve como salida un conjunto de datos producto de la mezcla de los dos que se proporcionaron como entrada al sistema. También podemos encontrar funcionalidad para representar gráficamente grafos y tener una percepción visual de los datos y relaciones que contiene. Uno de los módulos que forman la salida es un generador de imágenes que proporcionando dos nombres de conceptos, realiza búsquedas para seleccionar imágenes de estos y con ellas realiza un reconocimiento de los objetos que aparecen en esta. Con estos objetos de ambas imágenes realiza una mezcla y devuelve dicha imagen compuesta como salida. El otro módulo que representa el ouput del sistema es del que se habla en el capítulo 3 y que pretende ser remplazado en el futuro por la aplicación desarrollada al final de este trabajo.

Un ejemplo interesante de un módulo de extracción de información es el siguiente.

Texto del que se saca la información y a partir del cual se construyen las tripletas que representan el grafo semántico:

Easter, also called Resurrection Day, is a holiday. Among Christians, it is a celebration of Jesus Christ returning from the dead. Christians believe that it is the holiest day in the year. Some people who are not Christians celebrate it as the beginning of Spring. Easter is not held on the same date every year. This is called a moveable feast. Currently all Christian churches agree on how the date is calculated. Easter is celebrated on the first Sunday, following the first full moon, after the Spring Equinox. This means it is celebrated in March or April. It can occur as early as March 22 and as late as April 25. Western churches, like the Roman Catholic Church, use theGregorian calendar, while Eastern churches, like the Eastern Orthodox Church, use the Julian calendar.

Because of this the date of Easter celebrations is different for these two types of churches even though the way they calculate the date is the same. In 2015 Easter was celebrated on April 5 for both the Gregorian calendar and Julian calendar. The word .Eastërïs derived from Eastre, the name of the ancient German Goddess of Spring. Her festival occurred at the vernal equinox. The French word for Easter, Pâcques, comes from the Greek word for Passover, which is the Jewish holiday celebrated at the same time of the year. Jesus died on the cross about 2000 years ago in a city called Jerusalem (most of Jerusalem is in the modern country of Israel). The people who killed him did so because they believed that he was causing trouble for the government, and because he was claiming to be the Messiah. When they crucified him (meaning they nailed him to a cross), they even hung a sign over his head, which said, "King of the Jews."The day he was crucified is known by Christians as Good Friday. The New Testament states that on the Sunday after Jesus was killed, some of his followers found that his body was no longer in the tomb where he was laid. Later, Jesus is said to have appeared to over 500 people and preached to them. The New Testament teaches that the resurrection of Jesus is what Christianity is based on. The resurrection made people believe that Jesus was the powerful Son of God. It is also spoken of as proof that God will judge the world fairly. Christians believe that God has given Christians .a new birth into a living hope through the resurrection of Jesus Christ from the dead". Christians believe that through faith in God they are spiritually made alive with Jesus so that they may lead a new life.

Tripletas resultantes después del proceso de extracción de información del módulo:

(easter - isa - day)
(among_ christians - call - day)
(day - isa - holiday)
(among_ christians - call - holiday)
(among_ christians - be - celebration)
(roman_ catholic_ church - be - calendar)
(eastern_ church - use - calendar)
(eastern_ orthodox_ church - be - calendar)
(eastern_ orthodox_ church - use - calendar)
(name - isa - equinox)
(festival - occur_ at - equinox)
('pâcques'- isa - word)
('pâcques'- come_ from - word)
(king - be_ in - country)
(he - be_ claim_ be_ to - messiah)
(country - property - modern)
(king - be_ claim_ be_ to - country)

El código está alojado de forma privada en un repositorio *Git*. Podemos ver los aspectos técnicos para conocer el funcionamiento de esta plataforma de trabajo, cómo interaccionar con ella y cómo contribuir añadiendo recursos al proyecto en el apéndice A.

2.5.2. Contribución en la extracción de información

En este apartado vamos a mencionar otras contribuciones que se han hecho al proyecto. Nos vamos a centrar en aquellas que nos interesa estudiar para el desarrollo del sistema de este trabajo, como son la extracción de información y el filtrado y enriquecimiento del contenido. No se van a abordar fases anteriores a la generación de lenguaje natural, vamos a contar con que nuestra entrada ya ha pasado por estas anteriormente. Por esta razón, nos es interesante conocer las herramientas o aplicaciones que están a nuestra disposición para utilizar como entrada o mejorar esta.

La extracción de información es un proceso por el cual se extrae de una fuente no estructurada el conocimiento y se agrupa de una forma ordenada y coherente. Como ejemplo podemos consultar (Concepción et al., 2016). El proyecto *ConCrete* también tiene aportaciones de investigación en este área, en los cuales hemos contribuido, tanto publicando (Rodrıguez-Ferreira et al., 2016) como realizando un trabajo fin de máster (Rodríguez Ferreira, 2016).

El artículo con el que hemos contribuido a mejorar la fase de extracción de información se llama "Improving Information Extraction from Wikipedia Texts using Basic English", (ver apéndice B). Este explica que mediante el uso de recursos más apropiados se mejora la calidad y fiabilidad de la información extraída. El idioma y el registro utilizado en cada recurso influye en la forma de localizar el conocimiento que queremos extraer del texto. Nosotros comparamos varios recursos, la *English Wikipedia*[3], la *Simple English Wikipedia*[4] (también llamada *Simple Wikipedia*) y el *Simple English Wiktionary*[5]. Estos dos últimos recursos hacen uso del "Basic English", creado por (Ogden, 1944). Utilizando Freeling[6] (podemos probar la funcionalidad en la demo disponible en la web[7]), extraemos del texto su estructura sintáctica y reconociendo esta podemos seleccionar la información candidata a formar conocimiento, el cual representamos en forma de tripleta. Nos hemos centrado en reconocer y extraer únicamente la información que mantiene una relación

[3] `https://en.wikipedia.org`
[4] `https://simple.wikipedia.org`
[5] `https://simple.wiktionary.org`
[6] `http://nlp.lsi.upc.edu/freeling`
[7] `http://nlp.lsi.upc.edu/freeling/demo/demo.php`

Is o Is_A, es decir, descripciones (que representamos con el verbo en infinitivo Be), por ejemplo, *(chocolate - Be - dark)* o *(chocolate - Be - colour)*. Una vez tuvimos seleccionamos una cantidad representativa de conceptos de todo tipo, como objetos, emociones, lugares o personas entre otros, realizamos la búsqueda y extracción de información en cada uno de los recursos que seleccionamos y estudiamos anteriormente. Con todas las tripletas de cada recurso y concepto generamos una tabla con el contenido para estudiarlo y compararlo. Generamos los datos estadísticos que reflejan la cantidad de conceptos de los que se ha conseguido extraer conocimiento por cada recurso, así como la cantidad de conocimiento extraído. Pero la parte más interesante que estudiamos es la calidad del conocimiento extraído. Por ello necesitamos la ayuda de evaluadores humanos a los que pedimos valorar el conocimiento según un cierto criterio y con estos datos utilizamos una medida llamada "inter-annotator agreement", concretamente la de Fleiss Kappa (Fleiss et al., 2013). Con esta medida pudimos ver si estos evaluadores están más o menos de acuerdo en sus valoraciones, o si por el contrario la valoración de los datos no resulta objetiva, el resultado indicaba que existe cierta subjetividad a la hora de calificar los datos. Como conclusión del artículo y reflexión se explica que, si disponemos de un mecanismo sencillo de extracción de información podemos extraer de fuentes más grandes más conceptos y más conocimiento de cada uno, pero sacrificamos la su calidad. Si nos interesa más conseguir menos basura e información más fiable debemos elegir un recurso que utilice el idioma de una forma más simple. Gracias a este trabajo podemos utilizar la información para disponer de una fuente de conocimiento fiable aunque de tamaño reducido.

El trabajo fin de máster "Content Filtering and Enrichment Using Triplets for Text Generation" hace uso del artículo anteriormente mencionado. Éste trabajo abarca las áreas de extracción de información y determinación del contenido. Esta última es una fase de la generación de texto donde se decide qué información se va a contar y qué información se descarta.

Como entrada del sistema generado en el trabajo se toma información de ConceptNet 5[8], y de los tres recursos de los que se ha hablado en el artículo que se publicó (*English Wikipedia*, la *Simple Wikipedia* y el *Simple English Wiktionary*). También se hace uso de datos, concretamente un grafo semántico sobre un concepto, proporcionados por la universidad de Coimbra, que también son colaboradores en el proyecto. Como herramientas auxiliares se hace uso de *Freeling*, que proporciona la funcionalidad de analizador sintáctico y morfológico. El diseño de la aplicación final tiene las fases de parseo de la información, filtrado, enriquecimiento y parseo final, que dan lugar a la salida. La salida es un conjunto de tripletas que forman un grafo semántico.

[8] http://conceptnet5.media.mit.edu

La información contenida en la salida puede ser la de la entrada proporcionada por los colaboradores, en caso de que ésta tenga buena calidad y no sea necesario recortar ni aportar más contenido al grafo. También existe como posibilidad que la salida no contenga datos del grafo de entrada proporcionado, si estos datos no cumplen con una calidad suficiente. Sin embargo, lo que podemos esperar del sistema es una mezcla de dichos datos con datos extraídos de las otras fuentes mediante un proceso de extracción de información. Con esto se consigue que el grafo de salida tenga mayor cantidad de contenido a la vez que se aumenta la calidad de los datos en general.

Para enriquecer el grafo se utiliza el sistema desarrollado en el artículo (Rodrıguez-Ferreira et al., 2016). Este, como ya sabemos, proporciona según los recursos utilizados mayor o menor cantidad de información y calidad, por tanto debemos elegir la preferencia en cada caso. Durante la fase de enriquceimiento también se hace uso del recurso ConceptNet 5. El proceso que se sigue es realizar peticiones del concepto principal, y con los resultados volver a realizar peticiones para ver la similitud entre los datos extraídos de esta fuente y los datos de los que ya disponemos. Si comprobamos que esta similitud es suficientemente alta se incluye la información, de lo contrario se descarta.

La aplicación que se va a desarrollar con este trabajo no abarcará las fases de extracción de información y determinación del contenido. Como hemos podido observar, estas etapas son campos de la informática que tienen aún mucho trabajo de investigación. Nuestro sistema final tomará como entrada datos que ya hayan pasado por dichas fases, por ello, debemos tener en cuenta todo el trabajo realizado que se ha explicado en este apartado. Podemos explotar esta herramienta haciendo pasar nuestra entrada por ella y recogiendo la salida para trabajar con esta en vez de la original. Podemos conseguir esto ejecutando esta aplicación antes de la nuestra.

2.6. Recursos lingüísticos

Los recursos lingüísticos son fuentes de conocimiento estructurado que pueden ser utilizados de forma local o remota. Estos resultan necesarios para poder generar texto en lenguaje natural con una coherencia, sentido y significado. Por tanto, lo primero que buscaremos son recursos de donde sacar la información que buscamos y más adelante nos preocuparemos del programa, representación y tratamiento de los datos.

Comenzamos el estudio de recursos en internet para la extracción de información conceptual que necesitamos. Vamos a priorizar en la búsqueda

aquellos recursos que permitan la conversión de la información de forma sencilla a tripletas y a un grafo semántico, es decir, que no requieran la fase de extracción de información o esta no sea compleja. Debido a que existen infinidad de recursos actualmente y cada uno presenta ciertas propiedades y requiere ciertos requisitos para su uso, nos centraremos en algunos de los más conocidos y utilizados.

Los recursos que se han estudiado son: WordNet, ConceptNet, DBPedia y Thesaurus (secciones 2.6.1, 2.6.2, 2.6.3 y 2.6.4 respectivamente).

Existen también herramientas lingüísticas que pueden resultar de utilidad. Una de ellas es *Ultralingua*[9], esta proporciona datos sobre la conjugación de los verbos. Proporciona una *API* para poder solicitar los datos vía web. Se realiza la consulta proporcionando el idioma que se desea y el verbo en infinitivo. Como salida se obtienen los datos en formato JSON, para encontrar la información que se busca se tiene que recorrer la salida hasta que se encuentre la configuración que se requiere. Esto es porque los datos proporcionados son todas las formas verbales posibles, además de otras características como homónimos o pronunciación. Se puede ver una salida de ejemplo solicitandola en un navegador, en la dirección `http://api.ultralingua.com/api/conjugations/en/be`.

2.6.1. WordNet

WordNet es una herramienta que permite conocer muchas relaciones entre palabras. Se puede consultar en la web (WordNet Web (Miller, 1995) (Fellbaum, 1998) [10]) y utilizarlo, pero para integrarlo en un sistema, lo más sencillo es instalarlo y utilizarlo de forma local. Podemos observar la interfaz web que proporcionan en la figura 2.3. Las principales relaciones entre palabras que ofrece y que nos podrían interesar son las que explicamos a continuación.

- hyponym (direct y full): representa una lista de nombres que guardan la relación Es_Un (Is_A) con la palabra sobre la que se ha realizado la búsqueda. Si utilizamos los directos solo obtendremos los conceptos que cumplen la relación y son de la categoría inmediatamente siguiente a la consultada. De otra forma, si elegimos completa, obtendremos las palabras con esa relación sin tener en cuenta si existen categorías intermedias. Podemos conseguir el modo completo utilizanfo el modo

[9] `http://api.ultralingua.com/`

[10] `http://wordnetweb.princeton.edu/perl/webwn?o2=&o0=&o8=1&o1=&o7=&o5=&o9=&o6=&o3=&o4=&i=0&h=00#c`

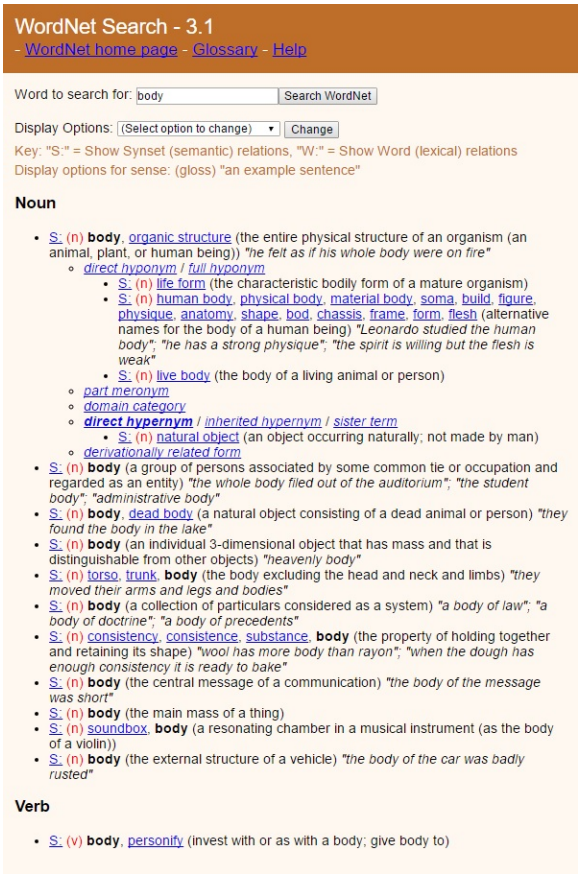


Figura 2.3: Interfaz web de WordNet

directo sobre todos los resultados que devuelva, así además podemos conocer la distancia que hay entre ambas palabras. Pongamos un ejemplo, gallina y loro son hipónimos de pájaro. En este caso podríamos extraer las tripletas (gallina - Es_Un - pájaro) y (loro - Es_Un - pájaro).

- part meronym: esta relación refleja el concepto Tiene (Have). Por ejemplo, un merónimo de pájaro es ala, por tanto la tripleta correspondiente sería (pájaro - Tiene - ala).

- member holonym: la holonimia permite representar la relación de hiponimia si se le da la vuelta, esto es, representa la relación Parte_De (Part_Of) o Miembro_De (Member_Of). Por tanto, podemos sacar la tripleta (brazo - Parte_De - cuerpo). O si preferimos unificar las relaciones de meronimia y holonimia podemos extraer la tripleta (cuerpo - Tiene - brazo).

- direct hypernym: esta relación también es la opuesta de otra que ya conocemos, el hipónimo. Podemos representarla como Pertenece_A (Belongs_To). Una tripleta de este tipo sería (hamster - Pertenece_A - roedor), si el concepto sobre el que hemos solicitado la relación es hamster y al menos uno de los resultados es roedor. Existe un modo, alternativo a la relación directa, la heredada (inherited). En este modo el resultado es una lista con todas las categorías superiores a las que pertenece, no solo la inmediatamente superior, que es la que devuelve el método directo.
- sister term: Esta relación muestra aquellas palabras que comparten las mismas categorías superiores, es decir pertenecen a lo mismo. Podría representarse esta relación como Similar_A (Similar_To). (hamster - Similar_A - ratón) es una tripleta que ejemplifica esta relación.

Este recurso necesita un pequeño proceso de extracción de información, no muy complejo, que aplica el conocimiento anteriormente explicado. La ventaja que ofrece este recurso es que proporciona información correcta. Como desventaja, las relaciones que se pueden extraer son muy limitadas.

2.6.2. ConceptNet 5

ConceptNet es un recurso que podemos consultar en la web ConceptNet [11]. Se trata de una fuente de información que principalmente proporciona tripletas. Podemos observar en la figura 2.4 los resultados que muestra como ejemplo la web de ConceptNet 5[12].

A continuación, en la tabla 2.1 se recogen las relaciones principales que se destacan en este recurso, pero dispone de muchas más. Esta lista puede consultarse en su web dedicada a ello[13], podemos encontrar estas relaciones junto con su significado.

Una forma muy sencilla de acceder a los datos de este recurso es por petición web ya que permite devolver una consulta con formato estructurado. El formato que sigue este recurso es JSON, en el que entre otra información, podemos encontrar las claves "start", "end" y "rel", que generarían la tripleta. Dentro de estos campos tenemos una cadena con los caracteres "/r" para relaciones y "/c" para conceptos. Después encontramos el idioma en el que está la información, por ejemplo "/en" para inglés. Las relaciones comienzan por mayúscula y las diferentes palabras se escriben juntas, comenzando

[11] `http://conceptnet5.media.mit.edu/`
[12] CC BY SA 3.0
[13] `https://github.com/commonsense/conceptnet5/wiki/Relations`

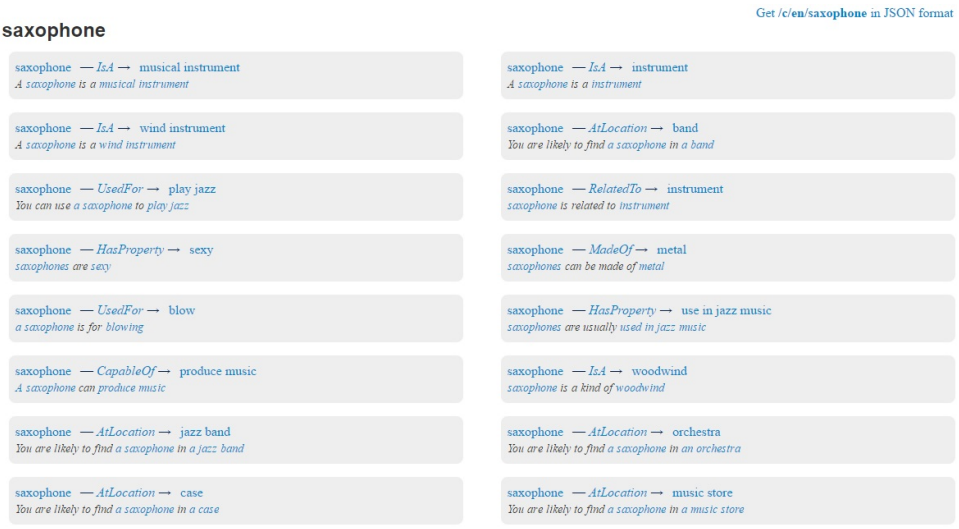


Figura 2.4: Resultados de ejemplo de la web de ConceptNet 5

/r/RelatedTo	*/r/IsA*	*/r/PartOf*
/r/MemberOf	*/r/HasA/r/HasA*	*/r/UsedFor*
/r/CapableOf	*/r/AtLocation*	*/r/Causes*
/r/HasSubevent	*/r/HasFirstSubevent*	*/r/HasLastSubevent*
/r/HasPrerequisite	*/r/HasProperty*	*/r/MotivatedByGoal*
/r/ObstructedBy	*/r/Desires*	*/r/CreatedBy*
/r/Synonym	*/r/Antonym*	*/r/DerivedFrom*
/r/TranslationOf	*/r/DefinedAs*	

Tabla 2.1: Principales relaciones en ConceptNet 5 expresadas con URI

siempre la primera letra con mayúscula. En el caso de los conceptos, las palabras son en minúscula y la separación entre palabras se representa mediante el carácter "_". En algunos casos, después de los conceptos o relaciones se concatena "/n" y se añade más información (normalmente para concretar o explicar). Esta última información la vamos a descartar de momento, ya que solo queremos una tripleta conceptual lo más simple posible. A continuación podemos ver un ejemplo:

... ''start'':''/c/en/golden_hamster/n/small_light-colored_hamster_often_kept_as_a_pet'', "rel": "/r/IsA", ''end'':''/c/en/hamster/n/short-tailed_old_world_burrowing_rodent_with_large_cheek_pouches'', ...

La tripleta generada correspondiente a la información mostrada en el ejemplo es (golden hamster - Is_A - hamster).

Si deseamos ver un archivo completo podemos consultar el apéndice C donde encontraremos la salida tal cual es proporcionada por este recurso. El concepto sobre el que se ha realizado la consulta es "actor". También podemos ver la información en: http://conceptnet5.media.mit.edu/data/5.3/c/en/actor/n/a_person_who_acts_and_gets_things_done.

2.6.3. DBpedia

La DBPedia es una fuente de conocimiento en la que se puede encontrar diferentes tipos de información. Podemos encontrar campos como *dbo:abstract* que contienen texto extenso, o *dbo:class* que contiene un enlace a otro concepto *dbr:Mammal*. Podemos consultar los datos en la web o podemos solicitar la respuesta a la consulta en otros formatos. En este recurso podemos encontrar numerosas pero limitadas relaciones, y bastantes de estas no nos resultan de utilidad, ya que son recursos como imágenes, referencias a otros recursos o información sobre estas referencias.

En la imagen 2.5 podemos apreciar un fragmento de los conceptos que nos puede proporcionar.

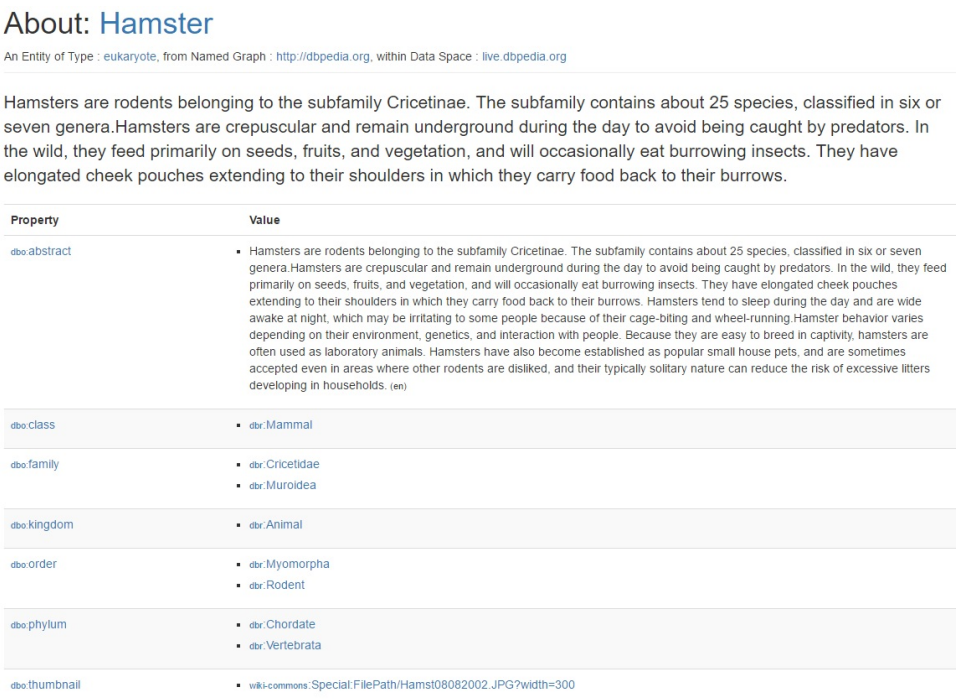


Figura 2.5: Ejemplo de consulta web de DBPedia

Las ventajas de este recurso son que permite obtener la información en varios formatos. Las desventajas son que en la interfaz web que proporcionan, la búsqueda resulta más tediosa y los datos que se proporcionan no son cuantiosos ni muy útiles. La información que se proporciona en forma de enlace es más compacta y más parecida a lo que buscamos. Podemos extraer del enlace el nombre del concepto al que se refiere, ya que es la última palabra o fragmento del enlace. Pero no podemos confiar plenamente en el fragmento de texto que aparece en un enlace para extraer estos importantes conceptos. La información de tipo texto que devuelve es un párrafo redactado por una persona y resulta complejo y difícil de manejar para realizar un proceso de extracción de información.

2.6.4. Thesaurus Rex

Existen muchos recursos que utilizan el término Thesaurus, pero en nuestro caso vamos a hablar de Thesaurus Rex[14], ya que la información es más sencilla de interpretar a la hora de hacer pruebas manualmente y los datos que proporciona son muchos y buenos. También podemos conseguir la información solicitando una versión en XML con la información, en este caso el tamaño de los conceptos se representa con un peso.

Como podemos observar en la imagen 2.6 los resultados de la búsqueda aparecen coloreados y en diferente tamaño.

En color verde podemos encontrar las categorías de grano fino del concepto de la búsqueda, representan la relación Es_Un (Is_A). Estos datos necesitan un procesamiento para mejorar el conocimiento y el contenido de la tripleta, ya que estas categorías tienen información ligeramente diferente (debido al matiz), por ejemplo, para *café* encontramos *public:facility* y *social:facility*. Pero, estas categorías están formadas por un matiz una categoría simple, lo que quiere decir que podríamos prescindir del resto de información proporcionada por el sitio sin perder datos.

En color rojo se indican los llamados matices que podemos representar mediante la relación Propiedad (Property). Esta información es más sencilla de extraer, no necesita procesar información extra como en el caso de las categorías, pero si queremos priorizar por importancia deberemos tener en cuenta el tamaño (peso en el XML). Para café, podríamos sacar la tripleta (cafe - Property - public), por ejemplo.

[14] `http://ngrams.ucd.ie/therex2`

En color azul tenemos las categorías simples del concepto, que se traducen en relaciones de tipo Es_Un (Is_A). La tripleta (cafe - Property - location) es un ejemplo para este caso.

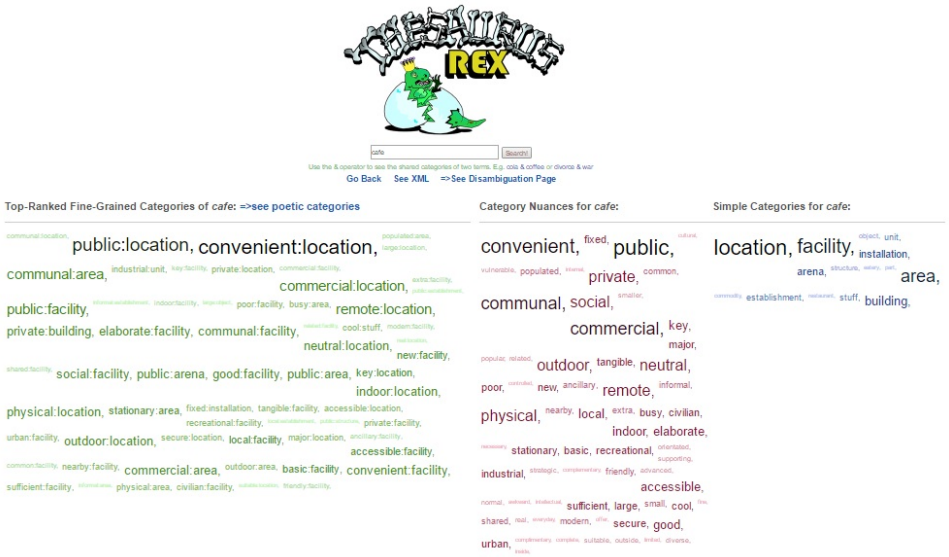


Figura 2.6: Información proporcionada por Thesaurus

Con este recurso podemos tener tripletas que representen Es_Un (Is_A) y Propiedad (Property) y podemos saber cuales de ellas se relacionan entre sí, conociendo mejor las relaciones sobre el concepto principal sobre el que se ha realizado la consulta. Podríamos representar el conocimiento que relaciona las diferentes relaciones con la relación Propiedad (Property). Por otro lado, son pocos tipos de relaciones los que podemos obtener.

Si deseamos ver un ejemplo del archivo *XML* que devuelve este recurso podemos hacerlo mirando el apéndice D de este trabajo. El concepto sobre el que se ha realizado la consulta es "cafe". También podemos consultarlo mediante la web, ya sea la vista diseñada para lectura por personas[15] o la vista del *XML*[16].

2.6.5. TextStorm

Este es un recurso especial, ya que está proporcionado por parte del proyecto *ConCreTe* para el que se desarrolla este trabajo, el cual ya hemos

[15] `http://ngrams.ucd.ie/therex2/common-nouns/member.action?member=cafe&kw=cafe&needDisamb=true`

[16] `http://ngrams.ucd.ie/therex2/common-nouns/member.action?member=cafe&kw=cafe&needDisamb=true&xml=true`

mencionado en 2.5 y no está disponible por el momento en internet. Es una herramienta de extracción de información con la que dado un texto se genera un grafo semántico, podemos ver más detalles en (Oliveira et al., 2001). Será accesible mediante la plataforma de trabajo del proyecto ConCreTe Flows explicada en 2.5.1, aunque por ahora trabajamos de forma local con los datos mediante archivos de prueba que nos han proporcionado.

Aunque vamos a estudiar y valorar este recurso, será uno de los que utilizaremos para enfocar y hacer las pruebas del generador de texto. La extensión del archivo de datos proporcionado es la de *prolog* (.pl) y los datos que se proporcionan son de la forma:

relación(concepto1, concepto2).

Podemos reconocer que en la sintaxis utilizada se utiliza como nombre del predicado el nombre de la relación, el cual siempre tiene dos argumentos, que conforman los conceptos a los que la relación se refiere. Y como es natural en prolog, cada linea contiene un predicado y termina en punto.

Este recurso nos da más datos útiles sobre el concepto principal sobre el que va a tratar el texto que los demás que hemos visto. No solo nos proporciona las tripletas del concepto sino que también nos facilita relaciones sobre los otros conceptos que aparecen en las tripletas. Gracias a esto podemos saber más detalles sobre la información de las relaciones y en definitiva sobre el concepto principal. Por ejemplo, *have(hamster, tail)* y *property(tail, short)* nos permiten realizar un texto más preciso e informativo.

En este recurso se utiliza una representación de los datos concreta, en la mayoría de los casos, cuando se requiere el uso de varias palabras para representar un concepto se utiliza el carácter "_" para unirlas. En algún caso excepcional se escriben las palabras juntas y en minúsculas (esto dificulta la extracción del conocimiento de forma automatizada), como en el caso de la relación "isa".

A continuación podemos ver un pequeño archivo de ejemplo (squirrel.pro), proporcionado por la universidad de Coimbra. El concepto principal de este grafo semántico es "squirrel".

```
live_in(kind, tree).
find(kind, nut).
eat(kind, seed).
isa(africa, australia).
have_be_introduce_to(africa, australia).
property(squirrel, black).
be(squirrel, black).
```

```
be(great_britain, black).
property(squirrel, common).
cause(squirrel, problem).
cause(great_britain, problem).
isa(raccoon, squirrel).
eat(hawk, squirrel).
eat(owl, squirrel).
look_after(mother, young).
can_have(squirrel, lifespan).
can_survive(some, '10').
```

2.6.6. Conclusiones de los recursos

El recurso que parece más sencillo para la extracción de tripletas y que contiene buena información, es decir, es correcta y precisa, es ConceptNet (2.6.2). Por tanto será en el que nos basemos para hacer la herramienta, además, como ya hemos dicho antes, utilizaremos el recurso de la universidad de Coimbra (2.6.5) con el que tenemos colaboración por el proyecto mencionado. Los otros recursos no los utilizaremos para sacar datos y probar la generación de textos, ya que no disponemos de tanto tiempo como para implementar la extracción de los datos y realizar tests y comparar los textos que salen de todos los recursos. Sin embargo, se podrá hacer uso de los recursos estudiados en el futuro, si alguno de estos nos puede proporcionar una funcionalidad que necesitemos.

Capítulo 3

Generación de texto a partir de información conceptual: *Textifier*

Textifier es una herramienta que se ha desarrollado con el fin de generar texto a partir de información conceptual. Esta herramienta forma parte de un sistema mayor que contribuye al proyecto *ConCreTe* (Concept Creation Technology) del que ya hemos hablado en la sección 2.5. Este sistema extrae información conceptual de dos conceptos dados y mezcla ambos dando lugar a un concepto nuevo. Este nuevo concepto es representado con imágenes y con texto, a modo de descripción y salida del sistema. Esta última tarea es la que aborda la aplicación sobre la que hablamos en este capítulo.

Dado que esta aplicación tiene que ser integrada en el proyecto mediante su plataforma, y ésta ya está definida, como hemos contado en 2.5.1, debemos adaptarnos a las exigencias que impone. Se pide integrar la aplicación como widget, y estos se programan en *Python*.

Por la parte del desarrollo del software, se prefiere la implementación en *Java*. Esto es debido a que se tiene más experiencia trabajando con este lenguaje, además de que también es un lenguaje multiplataforma y cuenta con muchas utilidades y librerías disponibles al alcance del programador.

La solución para satisfacer todas las restricciones anteriores es el desarrollo de la herramienta como servicio web. Resulta sencilla la implementación de una aplicación web en *Java* mediante un *IDE* como *Eclipse* con los paquetes necesarios para desarrolladores web. Esta aplicación se alojará en un servidor perteneciente a la Universidad Complutense de Madrid, en una má-

quina virtual que utiliza *Apache Tomcat*. Con esto, conseguimos integrar la aplicación en *ConCreTe Flows* mediante una petición web utilizando *Python*, como se solicita. Además, de esta manera, también podemos hacer accesible la aplicación para que se haga uso de ella de forma externa al proyecto, ya sea por parte de la universidad o toda la comunidad investigadora.

3.1. Entrada

Los datos de entrada al sistema son proporcionados por los colaboradores del proyecto, concretamente, por los del Instituto Jožef Stefan de Liubliana, Eslovenia. La entrada de la aplicación son datos representados en forma de grafo. Está construido en formato *JSON* con varios campos y características que vamos a explicar a continuación.

Este grafo siempre es dirigido, aunque tiene un campo en el que se indica si lo es o no. También tiene un campo que indica si se trata de un multigrafo. La característica principal que diferencia un grafo de un multigrafo es que en este último pueden aparecer aristas múltiples, también llamadas aristas paralelas. Este tipo de aristas representan varias relaciones entre una misma pareja de nodos, y pueden ser con identidad propia o sin ella[1]. Podemos ver las diferencias entre estos en la figura 3.1. En nuestro caso, el grafo tiene aristas sin identidad (las aristas no contienen información como un nodo) y no vamos a encontrarnos con casos de multigrafo. En caso de encontrarnos con uno es sencillo pasar del multigrafo a un grafo más simple, ya que las aristas no contienen información.

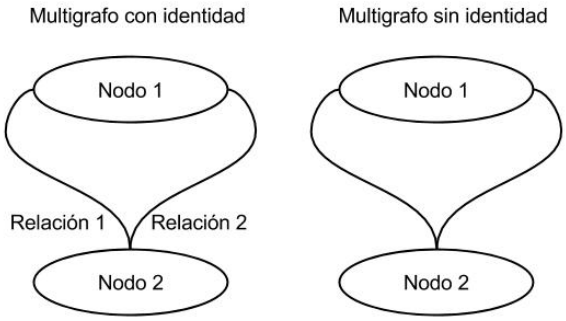


Figura 3.1: Diferencias entre multigrafos

Por otro lado, los campos más importantes y que conforman la entrada en sí reciben el nombre de *nodes* y de *links*.

[1]Información extraída de Wikipedia (`https://en.wikipedia.org/wiki/Multigraph`)

Por un lado, *nodes*, que es un campo de tipo *array*, guarda información conceptual de un único concepto. Este concepto representa indiferentemente un objeto o una relación. Estos nodos tienen dos campos, *id* y *label*. Ambos campos contienen la misma información en esta versión, pero se puede utilizar en el futuro para poder tener un mismo nodo con varios textos del concepto.

Por otro lado, *links* es otro campo de tipo *array*, este contiene objetos con los campos *source*, *target* y *key*. Los que nos interesan son *source*, que indica el índice del nodo que representa el principio de una relación entre nodos y *target*, que representa el final de la relación.

Por tanto, uniendo los nodos mediante estas relaciones podemos obtener uno o más grafos con esta entrada. Para obtener de esta estructura los datos en forma de tripleta, tendremos que recorrer el grafo e identificar el nodo o los nodos que representan cada una de las partes de la tripleta que construimos.

Los datos contenidos en los nodos son los llamados *conceptos*. Un concepto puede estar representado de muchas formas, en nuestro caso es texto, concretamente trozos o extractos de texto. Estos fragmentos de texto pueden o no ser los originales de la fuente de la que se extrajeron.

Con estos datos podemos construir tripletas. La forma de construirlas es tomando un nodo como concepto sujeto, otro nodo como relación y un último nodo como concepto objeto. Estos nodos tienen que cumplir que estén unidos mediante un link para que puedan tener sentido las tripletas resultantes. Debido a la estructura que tienen los datos no podemos asegurar la calidad de las tripletas, por ello realizaremos un procesamiento sobre toda la entrada e intentaremos mejorar la calidad de las tripletas resultantes.

Si se desea consultar un fichero de ejemplo que aceptaría esta herramienta, podemos ver el que aparece a continuación. Para mostrar el resultado de la construcción de los datos de este fichero en un grafo, podemos ver la figura 3.2. El proceso que se ha seguido para construirlo primero identifica los nodos del cero al n según su posición en la lista (en el ejemplo son del cero al trece, ya que hay catorce nodos). El siguiente paso es recorrer la lista de links, en cada link podemos establecer una conexión entre dos nodos. Creamos una lista de listas como estructura para almacenar los datos. Guardamos en la lista padre una lista que contiene el nodo origen y el final, si estos no estaban ya en alguna de las listas hijas. En caso contrario, para cada uno de estos nodos que coinciden con el nodo inicial o el final, se añadirá la relación con el otro. Será necesario duplicar aquellas listas donde el nodo que coincide no es el primer o último elemento de la lista. Se guardarán en la original los datos que ya estaban y en la nueva se sustituirán los elementos anteriores o posteriores según se tenga que insertar el nodo inicial o el final. De esta

forma conseguimos almacenar el árbol mediante una lista con los caminos a sus hojas.


```
{
   "directed": true,
   "graph": {},
   "nodes": [
      {"id": "are often used as", "label": "are often used as"},
      {"id": "be contains by", "label": "be contains by"},
      {"id": "rodents belonging to the subfamily Cricetinae", "label":
          "rodents belonging to the subfamily Cricetinae"},
      {"id": "contains", "label": "contains"},
      {"id": "have become established as", "label": "have become established
          as"},
      {"id": "about 25 species", "label": "about 25 species"},
      {"id": "the subfamily", "label": "the subfamily"},
      {"id": "about 25 species  classified  in  six  or  seven genera", "label":
          "about 25 species  classified  in  six  or  seven genera"},
      {"id": "hamster", "label": "hamster"},
      {"id": "they", "label": "they"},
      {"id": "popular small house pets", "label": "popular small house pets"},
      {"id": "be  classified  in", "label": "be  classified  in"},
      {"id": "laboratory animals", "label": "laboratory animals"},
      {"id": "six  or  seven genera", "label": "six  or  seven genera"}
      ],
   "links": [
      {"source": 0, "target": 12, "key": 0},
      {"source": 1, "target": 2, "key": 0},
      {"source": 3, "target": 7, "key": 0},
      {"source": 4, "target": 10, "key": 0},
      {"source": 5, "target": 11, "key": 0},
      {"source": 6, "target": 3, "key": 0},
      {"source": 7, "target": 1, "key": 0},
      {"source": 8, "target": 0, "key": 0},
      {"source": 9, "target": 4, "key": 0},
      {"source": 11, "target": 13, "key": 0}
      ],
   "multigraph": true
}
```


3.2. Arquitectura

El sistema está dividido en tres partes, la entrada, la salida y el procesamiento. En el módulo de la entrada se extraen los datos del recurso elegido

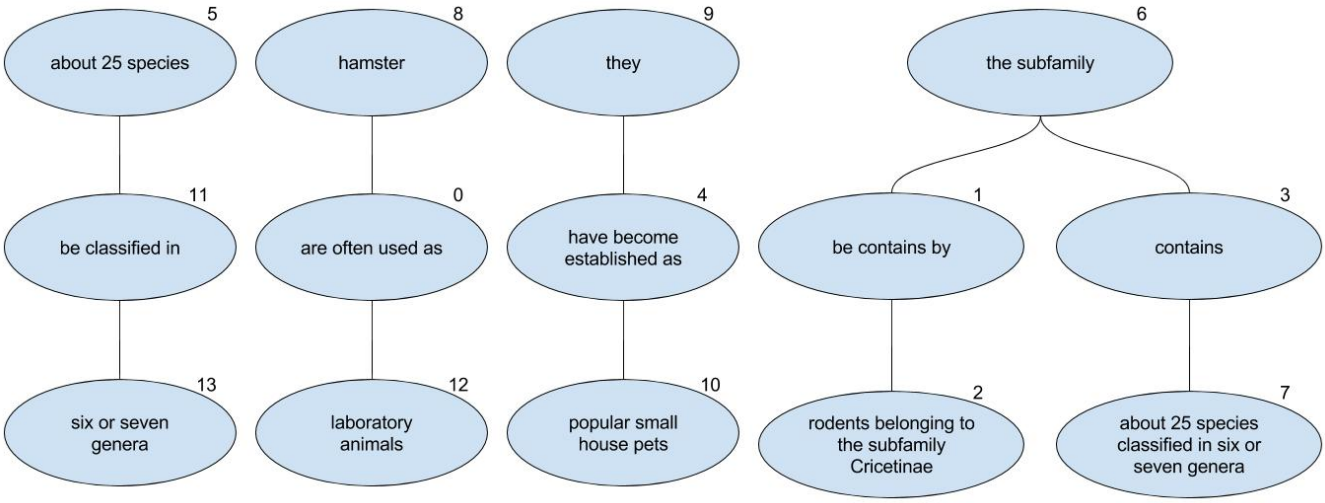


Figura 3.2: Grafo de los datos de entrada de Textifier construido

y se transforman al formato interno con el que se trabajará en la parte de procesamiento. En el módulo de salida se genera el texto según indica la estructura interna.

El formato interno que se utiliza para los datos es un árbol n-ario, donde el nodo raíz es vacío y solo contiene datos informativos y de control. Sus hijos son los nodos que no aparecen como nodo final (target) de una relación en los datos de entrada. De manera complementaria, los nodos hoja del árbol son aquellos que no aparecen como nodo inicial (source). Si nos fijamos en la figura 3.2, los nodos que vemos más arriba serían los hijos del nodo raíz, y los nodos que aparecen en la parte de abajo son los nodos hoja del árbol. A continuación podemos ver una representación del árbol que se generaría en el formato interno, tomando como datos de entrada los mismos que hemos visto anteriormente.

```
(root)
    |
    |___"about 25 species"
    |       |___"be classified in"
    |               |___"six orseven genera"
    |
    |___"hamster"
    |       |___"are often used as"
    |               |___"laboratory animals"
    |
    |___"they"
    |       |___"have become established as"
    |               |___"popular small house pets"
```

```
|
|___"the subfamily"
    |___"be contains by"
    |   |___"rodents belonging to the subfamily Cricetinae"
    |
    |___"contains"
        |___"about 25 species classified in six or seven genera"
```

El procesamiento que se realiza es una concatenación de funciones de módulos, donde el orden de ejecución de los módulos puede influir en el resultado. A continuación vamos a describir los módulos de los que disponemos actualmente y su funcionalidad.

- *TreeUnifier* es un módulo que permite coger los nodos hermanos que sean iguales y juntarlos en uno mismo que contiene como hijos los hijos de ambos nodos. Así conseguimos jerarquizar los conceptos según si están o no conectados a varios conceptos. Así se transforma una lista de listas de nodos en listas de árboles. A continuación podemos ver una transformación de este tipo.

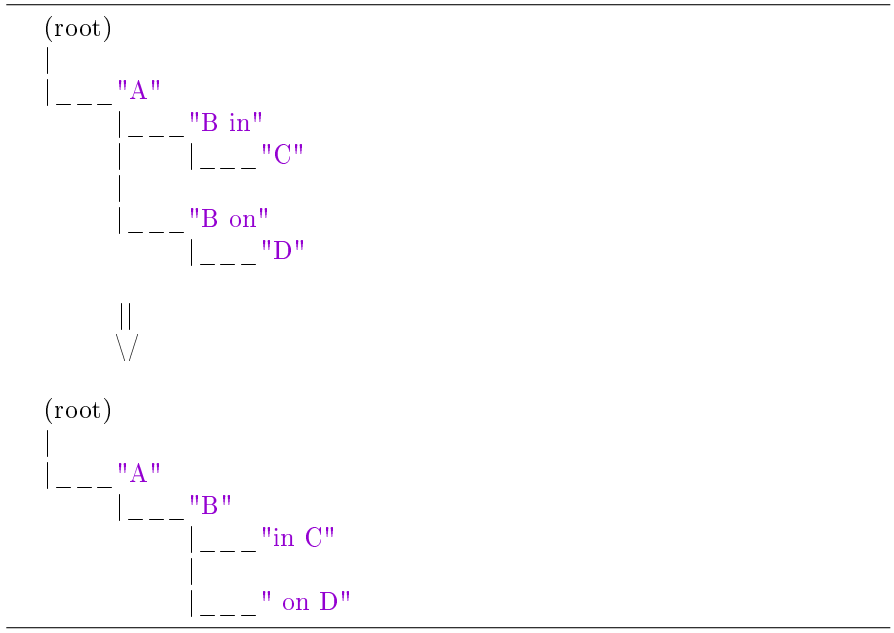
```
(root)
|
|___"A"
|   |___"B"
|       |___"C"
|
|___"A"
    |___"D"
        |___"C"

        ||
        \/

(root)
|
|___"A"
    |___"B"
    |   |___"C"
    |
    |___"D"
        |___"C"
```

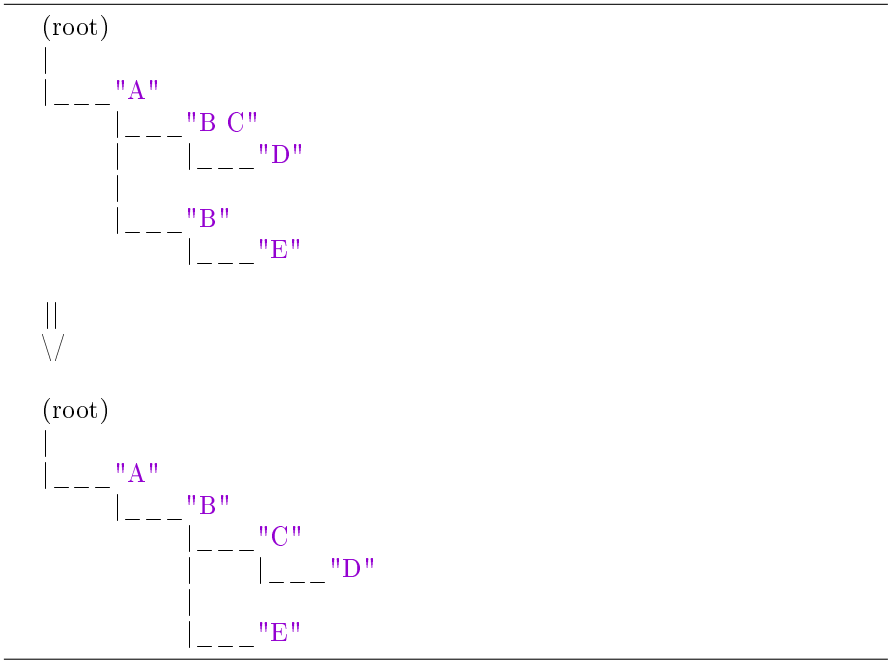
- El módulo *DifferentPrepositionUnifier* permite mejorar la calidad de los datos de entrada. El proceso que se sigue consiste en localizar varios nodos, generalmente se trata de nodos que representan relaciones.

Cuando se da este caso, donde dos nodos contienen información muy parecida, lo que se hace es construir un nodo con la información conceptual que tienen en común y añadir a los hijos la parte restante. De esta forma conseguimos una mejor estructura de los datos, que nos permite conocer mejor el tema que tratamos. El resultado es un nodo padre, que es el unificado, y dos hijos que heredan la preposición de la relación. Para clarificar la transformación podemos ver el esquema que aparece debajo.



- *DepthPruner* proporciona la funcionalidad de podar el árbol por su altura. Esto permite eliminar los trozos más largos del árbol. Esto tiene buenos resultados ya que los datos proporcionados como entrada contienen concentrado los mejores datos cerca de la raíz, y los proveedores recomiendan una poda a partir del tercer nivel, permitiendo reducir la basura y el ruido de los datos.

- También existe un módulo llamado *RepetitionUnifier* que encuentra los nodos que tienen parcialmente el contenido conceptual de otro nodo. La solución es que aquellos nodos con mayor contenido se convierten en un nodo con la información que no está ya representada en el otro nodo, además, se establece una relación padre-hijo entre ambos nodos, de forma que queda correctamente representada la misma información con la que empezamos. Esto ayuda a tener una mejor representación de los datos para poder ver de forma más clara las relaciones entre conceptos. El comportamiento de este módulo lo podemos ver ejemplificado en el que que aparece a continuación.



- *EntityExtractor* es el módulo que permite reconocer el concepto principal que relaciona la mayoría de los datos conceptuales proporcionados. Esto se consigue analizando y contabilizando todos los datos de los nodos de árbol. El resultado es una lista con los sustantivos y su frecuencia en los datos, y se almacena en la raíz del árbol ya que está designada para contener información de este tipo.

- *SentencePlanner* es un módulo que organiza la estructura de datos para que a la hora de generar el texto éste tenga un mejor orden. Su funcionalidad consiste en priorizar aquellos hijos de la raíz del árbol que contienen el concepto principal.

3.3. Implementación

El resultado de la generación de texto es diferente según cómo se concatenen los módulos. La secuencia de módulos que se ha elegido finalmente para describir los datos de entrada es la siguiente: *EntityExtractor*, *TreeUnifier*, *RepetitionUnifier*, *DifferentPrepositionUnifier*, *DepthPruner* (con el parámetro de poda a 3) y *SentencePlanner*. En la imagen 3.3 podemos ver el pipeline de estos módulos del sistema, desde la entrada hasta la salida. Esta disposición de los módulos se ha decidido debido a que la funcionalidad de estos resulta mejor si antes otros módulos se han ejecutado. Empezamos por

el *EntityExtractor*, ya que así podremos saber el concepto del que se habla si otro módulo posterior lo necesita. Los siguientes dos módulos que se ejecutan necesitan ir juntos, ya que el primero (*TreeUnifier*) junta las listas de nodos comunes para generar árboles y el segundo (*RepetitionUnifier*) utiliza estos árboles para detectar si se puede añadir al padre parte del contenido conceptual de sus hijos. Si no se ha unificado en árboles la estructura, el módulo *RepetitionUnifier* no detectará coincidencias. El módulo *DifferentPrepositionUnifier* también necesita la estructura de árbol para poder detectar que dos relaciones comparten parte de su contenido conceptual y se diferencian por una preposición, así puede unir la parte común y hacer que los hijos correspondientes hereden dicha preposición. Como podemos observar podríamos intercambiar el orden de los módulos *RepetitionUnifier* y *DifferentPrepositionUnifier*, ya que afectarán a nodos diferentes y no se influyen entre sí. Por último, se ejecuta la poda de los árboles con *DepthPruner*, esto se debe hacer en este momento, ya que hemos terminado de modificar la altura del árbol. Por último, cuando ya está preparada la estructura de datos para ser convertida en el texto final es el único momento en el que tiene sentido ejecutar *SentencePlanner*, de otra forma podría modificarse el orden y sería necesario ejecutarlo de nuevo.

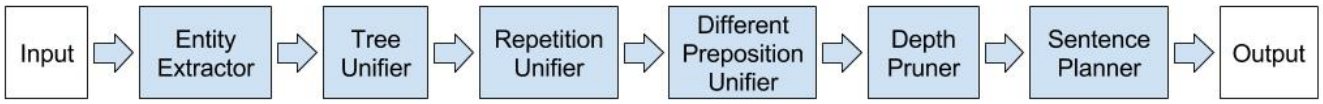


Figura 3.3: Pipeline del Textifier

De esta forma, primero obtenemos el concepto del que se va a hablar, luego creamos una estructura de árbol de forma que no hay nodos hermanos repetidos y jerarquizamos mejor la información. Más adelante separamos los nodos que contienen información adicional a otros nodos y unimos aquellos nodos que apenas se diferencian mejorando la estructuración de los datos. Finalmente, antes de organizar los datos para producir un texto más coherente se poda la información sobrante, como la basura o datos enrevesados que dificultan el entendimiento del texto (dejamos tres niveles del árbol, que representan una tripleta por cada hoja).

3.4. Salida

La salida del programa es texto plano contenido en un string. Funciona como servicio web, así que se le puede llamar desde cualquier sitio que disponga de conexión a internet. Un ejemplo de texto producido por esta herramienta es el que aparece a continuación.

Hamsters are rodents belonging to the subfamily Cricetinae. The subfamily contains about 25 species in six or seven genera. They have become established as popular small house pets. They are a bit like a mouse. Wild hamsters live in the desert, but people all over the world keep domesticated hamsters as pets. In the wild, hamsters are crepuscular and stay underground during the day. They feed on seeds, fruits, and vegetation, and occasionally eat burrowing insects. Hamsters are distinguished by their large cheek pouches, and relatively short tail. They use their long cheek pouches (extending to their shoulders) to carry food back to their burrows.

3.5. Conclusiones del *Textifier*

Si observamos la salida podemos ver que producimos un texto razonablemente bien estructurado y escrito. Además, parece natural, escrito por un humano. Esto es porque el contenido conceptual contiene texto complejo que ha sido redactado originalmente por una persona y después, el tratamiento que ha recibido no modifica o modifica muy poco el texto y su calidad.

Nuestro siguiente objetivo es generar texto con otro tipo de datos, datos conceptuales más simples, es decir, con texto menos complejo. Como la generación de texto depende mucho de la naturaleza y forma de los datos de entrada (representación del conocimiento), veremos que es necesario desarrollar desde cero otra aplicación para conseguir el mismo objetivo. Por tanto, cambiaremos la estructura de datos de entrada así como la interna y el procesamiento.

El tipo de datos que se quiere utilizar en el proyecto, al final de este, es un concepto representable con el mínimo número de palabras posible. Es decir, "popular small house pet" se puede considerar un concepto que indica una idea concreta. Pero, desde el punto de vista del proyecto, se encuentra más interesante el uso de varios conceptos y relaciones que representen el mismo conocimiento mediante el uso de conceptos más generales, y que se expresen con un menor número de palabras. Esto hace que tengamos un grafo de mayor tamaño, pero ayuda a conocerlo mejor por contener más relaciones. Gracias a la gran cantidad de vocabulario existente en inglés, y que este es el idioma más extendido y preferido en la comunidad de investigadores, vamos a tomarlo como el idioma para la representación de conocimiento. De esta forma la gran mayoría de los conceptos que tendremos representados en el sistema estarán representados con una única palabra, lo cual resulta óptimo para la preferencia del proyecto que hemos mencionado anteriormente. Por tanto, para el concepto "popular small house pet" podríamos construir el siguiente grafo semántico, con tripletas, que expresaría mejor la información:

(pet - be - popular)
(pet - be - small)
(pet - be - house)

Debemos denotar que esta conversión hace que se pierda el orden en el que aparecen los adjetivos de la idea original. Se podría reconstruir el orden si este originalmente sigue la conocida regla que sitúa primero los adjetivos más subjetivos y los más objetivos más pegados al sustantivo. Aunque para ello también deberíamos de disponer de un mecanismo para valorar los adjetivos y clasificarlos en una escala de objetividad. Por otro lado, el hecho de que no hay un orden impuesto nos permite que se pueda decidir cómo secuencializarlos según un criterio de énfasis.

3.6. Próximo sistema de generación de texto

Basándonos en las conclusiones (apartado 3.5), será necesario crear otro generador de texto para el nuevo formato y tipo de tripletas que se nos van a proporcionar. Por tanto, este sistema pretende reemplazar al *Textifier* y se hablará de él en el capítulo 5. Ya sabemos cómo son los datos que vamos a utilizar a partir de ahora en el proyecto *ConCreTe*, para adaptarnos a estos emplearemos una estructura de datos apropiada para ello, que nos permita trabajar de forma modular e independiente, conservando la idea del *Textifier*.

Ahora que sabemos las características de los recursos de información de los que vamos a partir para construir los grafos semánticos, que constituyen la entrada del sistema que queremos montar, podemos empezar a diseñar la aplicación.

Como los datos de los recursos que hemos estudiado y vamos a utilizar contienen la información en forma de tripletas o es sencillo transformar esta información a una tripleta, utilizaremos una lista de tripletas para recoger los datos de entrada. También tendremos una variable de tipo *String* que representa el concepto principal del que tratan los datos.

La estructura de datos *Triplet* que vamos a utilizar para representar una tripleta es muy simple. Tendremos tres campos, el primero es el campo que representa la parte derecha de la tripleta, que también llamaremos *concepto sujeto*. También almacenaremos en un *String* el nombre de la *relación*. Y finalmente, tendremos una *lista* de *Strings* donde almacenaremos las partes izquierdas de las tripletas o *conceptos objeto*.

El sistema tiene una fase fija, que es solicitar los datos a un recurso. En esta parte se extrae la información del recurso parseándola e introduciéndola

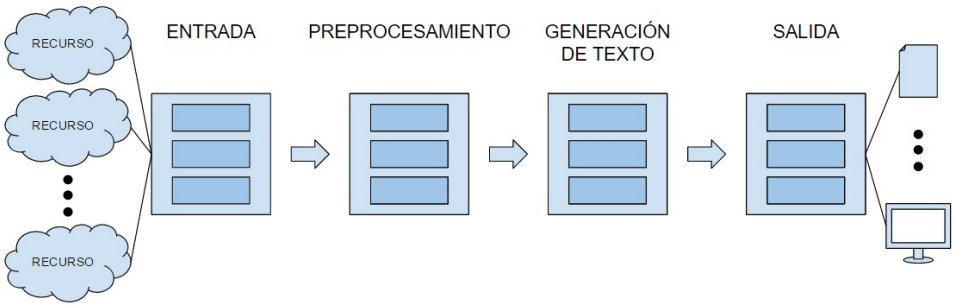


Figura 3.4: Arquitectura del sistema

en la estructura de datos que tenemos. Una vez hecho esto, dispondremos de varios módulos de preprocesamiento de los datos, que proporcionan utilidades como extraer el concepto del que hablan los datos o conocer estadísticas útiles de los datos. Después tendremos que elegir una de las formas de generar texto que queramos, estos serán módulos de generación de texto. Y finalmente tendremos la fase de salida, que por ahora será simplemente mostrar el texto generado por pantalla. Para ver la arquitectura del sistema podemos consultar la figura 3.4, donde veremos las fases que hemos mencionado de forma esquemática.

Capítulo 4

Procesamiento de la entrada

Como hemos mencionado antes, tendremos una serie de módulos que nos van a ayudar a conocer mejor los datos de entrada que nos llegan, y por tanto seremos capaces de realizar una mejor generación de texto. En este capítulo vamos a explicar cada uno de los que tenemos con un poco de detalle. Para situarnos mejor en el flujo de la aplicación y saber dónde nos encontramos ahora, podemos volver a la figura 3.4. Nos encontraríamos entre la fase de *Entrada* y la de *Preprocesamiento*, donde ya hemos solicitado y almacenado los datos de la fuente de conocimiento que hemos elegido.

Para entender mejor este capítulo, en el que vamos a mostrar código y a explicar el funcionamiento de los métodos, primero hablaremos de la estructura de datos que tenemos implementada en el sistema. Así entenderemos con mayor claridad cómo se consigue el resultado que queremos. Esta estructura nos permite almacenar el grafo semántico y operar con él de forma cómoda. Consiste en una lista ordenada de *Tripletas* que hemos modificado para poder sintetizar la información. Las tripletas de las que hemos hablado hasta ahora tienen el formato siguiente:

(Concepto1 - Relación - Concepto2)

En nuestro caso, hemos elegido una representación como la que aparece abajo:

(Concepto1 - Relación - [Concepto2, Concepto3, ...])

Como podemos observar en la estructura, conserva la parte izquierda de la tripleta y la relación de una tripleta convencional, el único cambio se encuentra en la parte derecha de la tripleta. Esta parte derecha, donde antes se almacenaba una cadena (*String*) con el *Concepto2* o como lo llamamos nosotros *concepto objeto*, ahora contiene una lista. Esta lista representa un conjunto de conceptos, más concretamente, representa la lista de conceptos

que comparten el mismo *Concepto1* o *concepto sujeto* y la *Relación*. Gracias a este cambio podemos disponer mucho más rápido (evitamos recorrer un grafo) de la información que guarda una misma relación con el concepto sujeto, lo que nos ayuda a realizar enumeraciones o encontrar la información que habla de lo mismo.

Esta nueva estructura permite la representación de las tripletas normales si lo deseamos, solo hay que mantener un concepto en cada lista, es decir, no agruparlos en una misma tripleta.

Ahora vamos a mostrar un ejemplo para que quede más clara la conversión que podemos tener en los diferentes formatos.

Formato de tripletas convencionales:

(cat - Is_A - animal)
(cat - Is_A - feline)
(cat - Is_A - predator)
(cat - Have - tail)
(cat - Have - claws)

Formato de tripletas nuevas conservando la representación de las antiguas:

(cat - Is_A - [animal)]
(cat - Is_A - [feline)]
(cat - Is_A - [predator])
(cat - Have - [tail])
(cat - Have - [claws])

Formato de tripletas nuevas que agrupan los conceptos:

(cat - Is_A - [animal, feline, predator])
(cat - Have - [tail, claws])

4.1. Main entity extractor

Este módulo toma el nombre de extractor de la entidad principal, ya que su objetivo es devolver el concepto principal de los datos. Para realizar su función necesita que le pasen la estructura con los datos y al recorrerlos almacena en un diccionario o mapa todos los conceptos sujeto que tenemos. Tenemos en cuenta que cada tripleta de nuestra estructura de datos puede contener varios conceptos objeto, lo que implica que se están representando varias tripletas reales. Finalmente, devolvemos el concepto sujeto que más veces ha aparecido.

La ejecución de esta funcionalidad no es estrictamente necesaria, por ejemplo, en caso de que se nos proporcione de antemano la entidad principal, el uso de esta utilidad no aportaría nada. Por otro lado, tenemos que decir que el algoritmo que se sigue se basa en que el grafo semántico que se proporciona tiene en la mayoría de sus tripletas el concepto principal del que se habla como concepto sujeto. En caso contrario, no se conseguiría extraer el verdadero concepto sobre el que trata la información proporcionada. Esto puede pasar con grafos grandes y detallados o grafos donde el proceso de extracción de información no ha tenido muy buen resultado y la información es de baja calidad.

Las tripletas que aparecen a continuación han sido extraídas con *TextStorm* y debido a los datos proporcionados, al ejecutar la utilidad que hemos construido, el concepto principal no coincide con el real. El concepto principal real sobre el que tratan los datos es "Easter".

Tripletas proporcionadas por *TextStorm*:

(easter - isa - day)
(among_ christians - call - day)
(day - isa - holiday)
(among_ christians - call - holiday)
(among_ christians - be - celebration)
(roman_ catholic_ church - be - calendar)
(eastern_ church - use - calendar)
(eastern_ orthodox_ church - be - calendar)
(eastern_ orthodox_ church - use - calendar)
(name - isa - equinox)
(festival - occur_ at - equinox)
('pâcques'- isa - word)
('pâcques'- come_ from - word)
(king - be_ in - country)
(he - be_ claim_ be_ to - messiah)
(country - property - modern)
(king - be_ claim_ be_ to - country)

Concepto principal extraído de la información de las tripletas por el sistema: Among christians

Para el resto de grafos semánticos que cumplen las expectativas que esperamos, que son la mayoría, esta funcionalidad da el resultado esperado y nos permite ubicar correctamente el concepto principal, como podemos ver en el siguiente ejemplo también sobre el concepto "Easter".

Tripletas porporcionadas por *ConceptNet*:

(easter - IsA - movable feast)

(easter - DefinedAs - celebration of resurection of christ)
(color egg - RelatedTo - easter)
(easter - RelatedTo - passover)
(pace egg - RelatedTo - easter)
(paske - RelatedTo - easter)
(pasch - IsA - easter)
(easter - Synonym - east wind)
(easter - DerivedFrom - eastre)

Concepto principal extraído de la información de las tripletas por el sistema: Easter

4.2. Statistics

Este módulo permite saber qué cantidad de tripletas hay y la cantidad de conceptos objeto que existen para una misma pareja de concepto sujeto y relación. Esto nos va a permitir conocer sobre qué tenemos más información y, probablemente lo que es más relevante en este tema. Podemos hacer sencillos cálculos con estos datos para saber mejor de qué tipo de información disponemos, como saber si la información esta más o menos enfocada a unas características o habla de muchas, la dispersión de la información. El cálculo es el total de tripletas del concepto principal divididas entre las diferentes relaciones para dicho concepto. Otro cálculo sencillo es dividir las tripletas que contienen la entidad principal entre todas las tripletas, lo que nos indicará la cantidad de detalles que se nos proporcionan de los conceptos que no tienen relación directa con el concepto principal.

Es necesario utilizar la estructura de datos que tenemos rellena para calcular los resultados. La utilidad se ha implementado utilizando un mapa donde la clave es el concepto sujeto y la relación, el valor asociado será la cantidad de tripletas que contienen los elementos de la clave pero entre ellos tienen diferentes conceptos objeto.

Para mostrar un ejemplo de la utilidad de esta funcionalidad hemos seleccionado los grafos semánticos de los recursos *TextStorm* y *ConceptNet* para el concepto principal "Squirrel". Si comparamos los resultados, vemos que el grafo de *ConceptNet* obtiene 9.4 en la dispersión de información, y 0.9591837 para la cantidad de detalles. Estos números nos revelan que este grafo tiene mucha información sobre cada relación y además esas relaciones hablan sobre la entidad principal, lo que hace de este grafo un buen candidato para generar un texto de calidad. Por otro lado, los resultados que muestran los datos extraídos del recurso *TextStorm* son, 1.25 para la dispersión, que se

traduce en que se tiene poca información de cada relación. Para la cantidad de detalles obtiene 0.29411766, lo que indica que dispone de más información adicional que el otro grafo. Si se desean ver más datos estadísticos sobre más conceptos sobre estos dos recursos se puede consultar la tabla 4.1.

Concepto	TextStorm dispersión	ConceptNet dispersión	TextStorm detalles	ConceptNet detalles
carnival	1.25	2.3333333	0.1724138	0.14
dolphin	1.5714285	3.909091	0.18644068	0.86
easter	1.5	2.1	0.1764706	0.5675676
hamster	1.6666666	1.9285715	0.3125	0.9310345
squirrel	1.25	9.4	0.29411766	0.9591837
whale	1.6	1.4242424	0.10526316	0.94

Tabla 4.1: Tabla de dispersión y calidad de los datos en varios recursos y conceptos

4.3. Relation simplifier

Esta utilidad coge la estructura de datos y mediante una tabla de traducción cambia las relaciones por otras para simplificar la información. Por ejemplo, en ciertas fuentes aparecen las relaciones Es (Is), ES_Un (Is_A) y Es (Be) de forma indiferente. Esto es un problema que se obtiene por el uso de recursos de diferente naturaleza, y se genera al inferirse desde la fase de extracción de información que se ha llevado a cabo por cada uno de los recursos, estos los almacenan según su criterio. Por ello la información puede no expresarse igual y dar este problema. Comúnmente en tareas relacionadas con la generación de información se etiquetan como Es (Is) o Es_Un (Is_A) las relaciones que aparecen en el texto como el verbo ser (to be) conjugado en cualquiera de sus formas. También podemos encontrar la traducción de Es (Is) o Tiene (Has) como Propiedad (Property), pero esta puede llevar a confusión en algunos casos debido a que no podemos saber con certeza cuál si tenemos que realizar el cambio de Propiedad (Property) a Es (Is) o a Tiene (Has), por tanto omitiremos su sustitución. Esta también es una forma sencilla de enmendar errores detectados de palabras mal escritas. Una solución es estudiar detenidamente los recursos y generar una tabla de traducción interna para poder unificar como deseamos las relaciones, pero resulta una tarea poco práctica.

Teniendo en cuenta lo mencionado anteriormente, se ha creado una tabla a mano donde se simplifican las relaciones que aparecen de diferente forma pero contienen un significado conceptual idéntico, ésta es la tabla 4.2. Esta tabla se puede ir actualizando a medida que se realizan test y pruebas so-

bre los datos de las diferentes fuentes, pero desde un principio contiene las traducciones que se conocen.

Relación	Traducción
isa	be
is a	be
is	be
feed	eat
fee	eat
feed on	eat
fee on	eat

Tabla 4.2: Tabla de traducción de relaciones

Lo que se consigue con este proceso es eliminar las relaciones que pueden resultar molestas a la hora de generar un texto coherente y poco repetitivo, ya que el generador tomaría las relaciones como si su significado fuera diferente. Pondremos un par de ejemplos con el formato interno del sistema, para compararlos y ver más claras las ventajas. La entrada para ambas generaciones es la siguiente.

Tripletas de entrada:

(frog - Is - green)
(frog - Is_A - amphibian)
(frog - Be - poisonous)

En el primero que aparece vemos que las frases se construyen correctamente, pero no se encuentra la relación entre ellas, ya que son relaciones con representaciones diferentes (Is, Is_A y Be) aunque semánticamente semejantes, pero esto último no lo sabemos.

Tripletas que entran al módulo de generación (sin simplificar):

(frog - Is - [green])
(frog - Is_A - [amphibian])
(frog - Be - [poisonous])

Texto de salida:

A frog is an amphibian. They are green. Frogs are poisonous.

En el segundo y último ejemplo, podemos ver que las relaciones ahora tienen la misma representación, lo que implica que la aplicación sabe que son semánticamente iguales y por tanto se podrían mezclar los conceptos mejor.

Tripletas que entran al módulo de generación (simplificadas):

(frog - Be - [green])
(frog - Be - [amphibian])
(frog - Be - [poisonous])

Texto de salida:

Frogs are green poisonus amphibians.

En la figura 4.1 podemos ver el grafo de las tripletas de la tabla 4.3, que representan un conjunto de datos de ejemplo del recurso TextStorm (descrito en el apartado 2.6.5), el concepto principal en este caso es "hamster". Para ver el resultado de la ejecución de la utilidad del módulo *Relation simplifier* podemos ver la figura 4.2, donde aparecen los cambios en las relaciones (resaltados en rojo), y en la tabla 4.4 las tripletas resultantes (también marcadas en rojo).

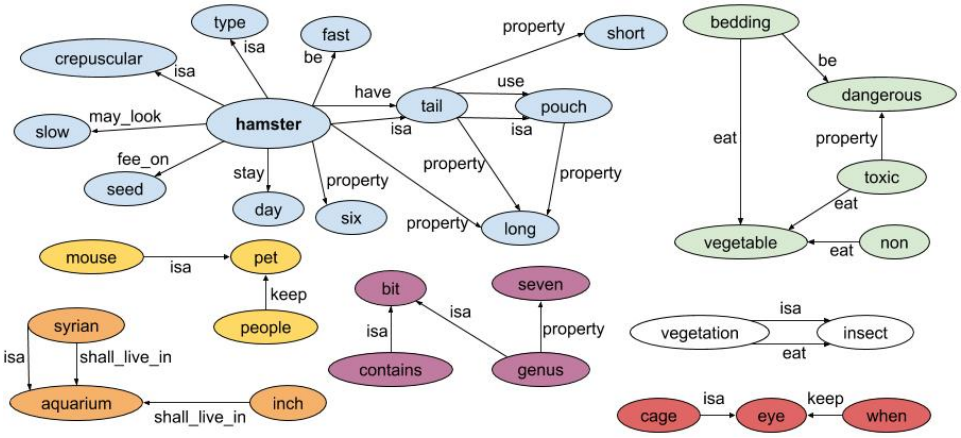


Figura 4.1: Grafo semántico sobre "hamster" extraído por TextStorm

4.4. Relation unifier

Este es el módulo que permite almacenar las tripletas de forma que en vez de encontrar un único concepto objeto en la parte derecha de la tripleta encontramos una lista de conceptos objeto. Así tenemos representado el conocimiento que se refiere a lo mismo junto, conocemos mejor los datos y podemos realizar fácilmente enumeraciones.

Para conseguir esto creamos una nueva lista de tripletas que es la que devolveremos para sustituir a la vieja. Utilizaremos un mapa donde la clave será la pareja formada por el concepto sujeto y la relación y el valor será el índice de la lista donde se encuentra esa tripleta. De esta manera recorremos

(contains - isa - bit)	*(genus - property - seven)*
(genus - isa - bit)	*(mouse - isa - pet)*
(people - keep - pet)	*(hamster - isa - crepuscular)*
(hamster - stay - day)	*(hamster - fee_on - seed)*
(vegetation - isa - insect)	*(vegetation - eat - insect)*
(tail - property - short)	*(tail - isa - pouch)*
(pouch - property - long)	*(tail - use - pouch)*
(hamster - property - six)	*(hamster - isa - type)*
(hamster - property - long)	*(hamster - isa - tail)*
(tail - property - long)	*(hamster - have - tail)*
(toxic - property - dangerous)	*(bedding - be - dangerous)*
(non - eat - vegetable)	*(toxic - eat - vegetable)*
(bedding - eat - vegetable)	*(syrian - isa - aquarium)*
(inch - shall_live_in - aquarium)	*(syrian - shall_live_in - aquarium)*
(cage - isa - eye)	*(when - keep - eye)*
(hamster - may_look - slow)	*(hamster - be - fast)*

Tabla 4.3: Tripletas sobre "hamster" extraídas por TextStorm

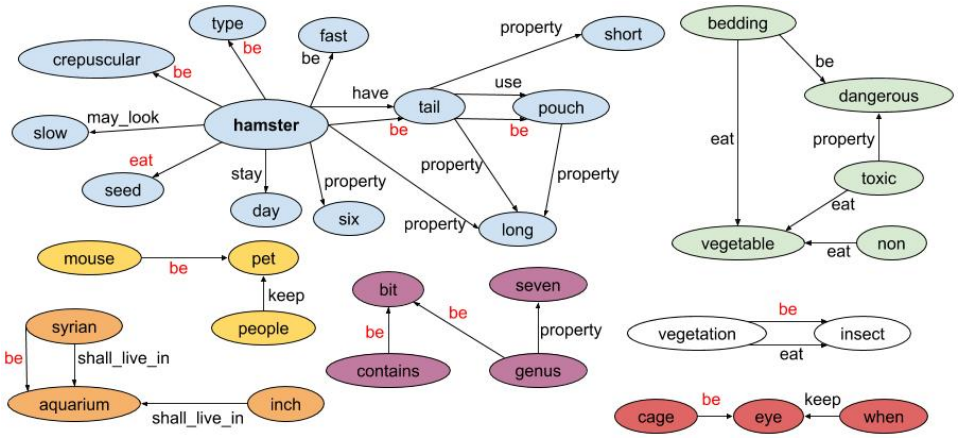


Figura 4.2: Grafo semántico sobre "hamster" extraído por TextStorm con las relaciones simplificadas

las tripletas antiguas que se nos proporcionan y si existe la clave en el mapa añadiremos el concepto objeto correspondiente a la tripleta que se encuentre en la clave contenida en el mapa.

Para ver un ejemplo de forma gráfica podemos consultar el grafo de la figura 4.3. Este se ha conseguido a partir del grafo de la figura 4.2 (tripletas 4.4) al ser procesado por el módulo *Relation unifier*. Adicionalmente podemos consultar las tripletas resultantes del proceso con el formato interno del sistema en la tabla 4.5.

(contains - be - bit)	*(genus - property - seven)*
(genus - be - bit)	*(mouse - be - pet)*
(people - keep - pet)	*(hamster - be - crepuscular)*
(hamster - stay - day)	*(hamster - eat - seed)*
(vegetation - be - insect)	*(vegetation - eat - insect)*
(tail - property - short)	*(tail - be - pouch)*
(pouch - property - long)	*(tail - use - pouch)*
(hamster - property - six)	*(hamster - be - type)*
(hamster - property - long)	*(hamster - be - tail)*
(tail - property - long)	*(hamster - have - tail)*
(toxic - property - dangerous)	*(bedding - be - dangerous)*
(non - eat - vegetable)	*(toxic - eat - vegetable)*
(bedding - eat - vegetable)	*(syrian - be - aquarium)*
(inch - shall_live_in - aquarium)	*(syrian - shall_live_in - aquarium)*
(cage - be - eye)	*(when - keep - eye)*
(hamster - may_look - slow)	*(hamster - be - fast)*

Tabla 4.4: Tripletas sobre "hamster" extraídas por TextStorm con las relaciones simplificadas

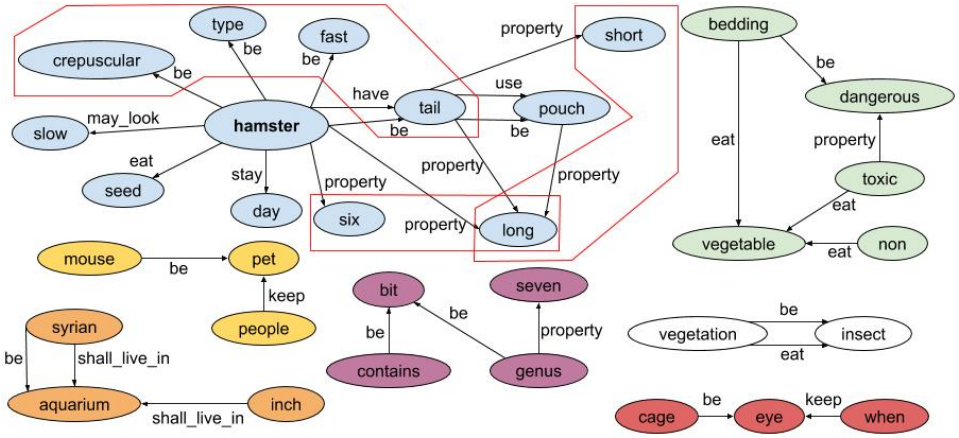


Figura 4.3: Grafo semántico sobre "hamster" extraído por TextStorm con las relaciones simplificadas y unificadas

4.5. Relation sequencer

Para utilizar este módulo debemos proporcionar las tripletas y el concepto principal del que hablan. En caso de que este último valor se proporcione como nulo (*NULL*), se ordenarán las tripletas de forma que aparezcan primero aquellas tripletas que contengan más conceptos objeto. Por tanto quedará ordenado de mayor a menor cantidad de información. Se supone que tiene

(hamster - be - [crepuscular, type, tail, fast])	*(hamster - property - [six, long])*
(hamster - stay - [day])	*(hamster - eat - [seed])*
(hamster - have - [tail])	*(hamster - may look - [slow])*
(tail - property - [short, long])	*(contains - be - [bit])*
(genus - property - [seven])	*(genus - be - [bit])*
(mouse - be - [pet])	*(people - keep - [pet])*
(vegetation - be - [insect])	*(vegetation - eat - [insect])*
(tail - be - [pouch])	*(pouch - property - [long])*
(tail - use - [pouch])	*(toxic - property - [dangerous])*
(bedding - be - [dangerous])	*(non - eat - [vegetable])*
(toxic - eat - [vegetable])	*(bedding - eat - [vegetable])*
(syrian - be - [aquarium])	*(inch - shall live in - [aquarium])*
(syrian - shall live in - [aquarium])	*(cage - be - [eye])*
(when - keep - [eye])	

Tabla 4.5: Tripletas unificadas con el formato interno del sistema

más relevancia aquella información de la que se dispone más cantidad de datos.

Por otro lado, si el concepto principal no es nulo, creamos dos listas, una para aquellas tripletas que contengan (como concepto sujeto) el concepto principal y otra para el resto de las tripletas. A continuación ordenamos de mayor a menor cantidad de información las tripletas de ambas listas. Y finalmente, unimos las listas en una sola, dejando en primer lugar la lista que contiene el concepto principal como concepto sujeto y al final la lista con el resto de tripletas ordenadas. De esta forma la información que queda primero es la que está directamente relacionada con el término principal y de la que más información se dispone.

Al tener la información ordenada podemos decir que tenemos una estructuración de las ideas. Esta organización se basa en las relaciones, tomamos como más importante el concepto del que se va a hablar, ya que no tendría sentido de otro modo. Las relaciones indican a su vez conceptos, entonces, las ideas más relevantes serán las que unen el concepto principal, la relación que contiene más conceptos y esos mismos conceptos. Así es como hemos decidido crear y organizar las ideas en el texto, de mayor a menor relevancia. Podríamos secuenciar las tripletas sobre los conceptos objeto después de nombrar estos, pero es tarea del generador utilizar esta información como adjetivo de dicho concepto objeto, y no crear una frase con la tripleta que especifica la característica. Por esta razón no nos preocupamos de la secuenciación de estas tripletas, además, de esta forma nos ahorramos el tiempo de

procesamiento que requeriría la búsqueda del concepto sujeto como concepto objeto en cada tirpleta.

Ahora vamos a mostrar un ejemplo de ordenación, el grafo de entrada para este caso es el siguiente, que vamos a suponer que ha sido modificado por las funcionalidades ya explicadas *Relation simplifier* (sección 4.3) y *Relation unifier* (sección 4.4), en el orden mencionado. Con esto veremos que hemos conseguido secuenciar las ideas como queremos y disponemos de este orden para que en la etapa de generación de texto solo tengamos que recorrer de principio a fin los datos.

Grafo de entrada:

(dolphin - capable_ of - [swim, bite, jump])
(mammal - property - [intelligent])
(dolphin - live_ in - [estuary, ocean])
(cetacean - have - [eye])
(dolphin - be - [carnivore, aquatic, mammal, intelligent])
(animal - have - [eye])
(dolphin - hear - [underwater])

Grafo obtenido:

(dolphin - be - [carnivore, aquatic, mammal, intelligent])
(dolphin - capable_ of - [swim, bite, jump])
(dolphin - live_ in - [estuary, ocean])
(dolphin - hear - [underwater])
(mammal - property - [intelligent])
(cetacean - have - [eye])
(animal - have - [eye])

Esta funcionalidad, el *Relation sequencer*, es el último de los módulos que nos permiten preparar los datos de entrada y acomodarlos a nuestras necesidades, para poder generar con más facilidad y comodidad el texto. El orden de ejecución en que recomendamos aplicar las funcionalidades explicadas en este capítulo es el mismo en el que se han mencionado: *Main entity extractor*, *Relation simplifier*, *Relation unifier* y *Relation sequencer*. En este punto el grafo semántico está representado con el formato interno del sistema y está preparado para la fase de generación de texto, si volvemos a la figura 3.4, nos encontramos a la entrada del módulo *Generación de Texto*, último paso antes de obtener la salida.

Capítulo 5

Generación de texto

En este capítulo vamos a explicar las diferentes formas que hemos elegido para generar el texto a partir de los datos que nos llegan de la fase de preprocesado, que hemos explicado en la sección 4. Esta es la última fase del sistema antes de generar la salida. Existen muchas formas de generar lenguaje natural, y cada uno de los generadores tiene unas características en cuanto a complejidad de programación, arquitectura, extensibilidad y calidad del texto. Estudiaremos más a fondo aquellas formas que hemos creado y desarrollado.

Vamos a explicar cronológicamente los generadores de texto que hemos implementado, desde el primero y más simple al último. Para poder mejorar y crear una nueva versión haremos una crítica, y con ella procuraremos que el siguiente modelo elimine las carencias del anterior intentando mantener al menos el resto de las características. Por último, al final de este capítulo encontraremos una reflexión final que muestra los resultados con una discusión crítica y razonada de los mismos. Esta consistirá en un resumen y una comparación de lo más destacable de los generadores que hemos construido.

5.1. Generando texto de forma básica

Al primer generador de textos que se va a crear se le ha asignado el nombre de *generación simple de texto*. El proceso que se sigue para para conseguirlo comienza colocando el texto del concepto sujeto, luego se añade el texto de la relación y finalmente el texto del concepto objeto. Para seguir las reglas de la lengua, se escribe la primera letra en mayúscula y se coloca punto al final.

Para unir en un mismo texto las tripletas del mismo concepto se puede crear una frase por cada tripleta, de esta forma queda un texto muy sencillo con una idea por línea. A continuación podemos ver un ejemplo.

Grafo semántico de entrada:

(hamster - Be - mammal)
(hamster - Be - rodent)
(hamster - Be - animal)

Texto de salida:

Hamster be mammal. Hamster be rodent. Hamster be animal.

Si queremos realizar oraciones más complejas podemos utilizar enumeraciones en aquellas relaciones que compartan el mismo concepto sujeto. Separaremos con el carácter coma (“,”) los conceptos objeto y entre los dos últimos un conector (en inglés “and”). El algoritmo desarrollado en este trabajo utiliza por defecto esta última forma para generar el texto, usando enumeraciones. En el siguiente ejemplo podemos ver la salida en forma de enumeración, los datos de entrada son los mismos que en el ejemplo anterior, así podemos comparar ambas salidas.

Texto de salida como enumeración:

Hamster be mammal, rodent and animal.

Como las frases se generan por tripletas y no se relacionan las ideas de unas tripletas con otras, para realizar un texto coherente se eliminan todas aquellas que no tienen sentido mencionar. Por este motivo no se genera texto con las tripletas que no contienen el concepto principal como concepto sujeto.

Con un ejemplo de texto generado por éste método podemos ver los cambios y transformaciones más claramente y resulta más visual. El ejemplo es el siguiente:

El grafo semántico correspondiente a la entrada se puede consultar en la tabla 4.4 (que ya hemos visto en un capítulo anterior) para mostrar la información de las tripletas que lo conforman. También podemos consultar la figura 4.3, que es la representación visual de esta información.

Texto de salida:

Hamster be crepuscular, type, tail and fast. Hamster property six and long. Hamster stay day. Hamster eat seed. Hamster have tail. Hamster may look slow.

Como hemos podido observar en los ejemplos, en esta forma primitiva de generar texto no se comprueba que la forma verbal sea la correcta para la frase y tampoco se añaden elementos adicionales a la información que aparece estrictamente en el grafo de conocimiento. En este caso, si nos fijamos en la figura 4.3, podemos comprobar que la única información que aparece en el texto es la que tiene como origen el nodo "hamster" que aparece en negrita en el grafo coloreado de azul, sin tener en cuenta la relación o el nodo final con el que se une.

Otra observación sobre el texto resultante que hemos visto es que la salida conseguida con esta generación de texto es muy dependiente de los datos de entrada. Contamos con que la información proporcionada desde fuera sea correcta, pero la representación de las relaciones como texto es muy importante para la adecuada generación de texto.

Si partiésemos de los datos siguientes.

(hamster - Be - cepuscular)
(hamster - Be - rodent)
(hamster - Be - mammal)
(hamster - Be - fast)
(hamster - Eat - seeds)
(hamster - Have - tail)
(hamster - Have - legs)
(hamster - Seem - slow)

La salida generada será la que aparece a continuación. Esta se considera algo mejor, ya que los datos son más correctos, solo presenta relaciones que son verbos y no contiene verbos modales.

Hamster be crepuscular, rodent, mammal and fast. Hamster have tail and legs. Hamster eat seeds. Hamster seem slow.

Podemos añadir en el código una serie de sencillas comprobaciones a mano donde, para las relaciones más comunes o conocidas, se genere mejor el texto. Estas son como las que hay en la tabla de traducción del módulo de preprocesamiento que tenemos y del que hablamos en la sección 4.3. Para el ejemplo anterior y como se muestra en los textos de salida, podríamos añadir la conversión de la relación "May_look" por "Seem" para mejorar la generación. La ventaja que da esto es que el texto resultante es de mayor calidad ya que es más natural. Por otro lado, las desventajas son que puede ser poco frecuente que se consiga ejecutar el caso o por el contrario repetitivo si hay varias. Además, como hemos dicho, es necesario realizar a mano los casos y se convierte en una tarea con poco rendimiento en cuanto al tiempo empleado.

Como reflexión final de este apartado, hay que decir que esta es una forma muy sencilla de generar texto, no tiene algoritmos complejos de programar y se puede crear en un solo módulo. No existe necesidad de crear un flujo de módulos o *pipeline* para ejecutar varias funcionalidades en cierto orden. Permite como hemos visto mejorar su funcionamiento a base de añadir casos concretos y esto es sencillo de programar también, aunque puede no resultar práctico generar una cantidad grande de casos, ya que haría el código menos legible y mucho más extenso. La calidad del texto no es muy alta, ya que las frases generadas son oraciones de tres palabras o enumeraciones con el verbo sin conjugar y sin elementos adicionales como preposiciones necesarias.

5.2. Estudio de textos generados por usuarios

Si queremos conseguir generar lenguaje natural que parezca redactado por un humano tendremos que estudiar textos redactados por humanos. Para fijarnos en el tipo de textos que suenan más naturales vamos a analizar textos generados por usuarios y valorar sus características. El experimento consistirá en proporcionar los mismos datos de los que nosotros disponemos y pedirles que escriban un texto coherente. Deberán tener en cuenta que no pueden utilizar información que no se les proporcione en los datos del enunciado.

Debido a la naturaleza de los datos que se van a facilitar, los usuarios que participen deben conocer la estructura de datos y deberán poder interpretarla, por ello tendrán que tener conocimientos sobre informática o un guía que les ayude. Se les proporcionará esta información para simular un generador de texto, ya que si les facilitamos los datos cambiados para que los entienda cualquier usuario podríamos estar influenciando su respuesta. Este requisito hace que sea más difícil encontrar candidatos. Por otro lado, intentaremos tener un número limitado de respuestas, ya que es necesario realizar un análisis de las características a mano, tarea que consume mucho tiempo. El objetivo de analizar los textos a fondo es extraer las características más comunes entre los textos que demuestran signos de naturalidad y coherencia entre los datos.

El experimento consta de dos cuestionarios independientes. La única diferencia entre ellos es la cantidad de tripletas proporcionadas. El objetivo de tener varios cuestionarios es para obtener datos de textos más y menos detallados y complejos, así podremos crear una plantilla mejor para todos los casos.

La información proporcionada (tripletas) han sido extraídas de: *TextStorm* (apartado 2.6.5) y *ConceptNet 5* (apartado 2.6.2). Hemos eliminado los datos manualmente que hemos considerado basura, ya que aún no se ha automatizado su eliminación y este trabajo no abarca ese aspecto. Estos datos que se han quitado son aquellos que contienen información errónea o que no guardan relación con el concepto del que se habla. Algunas tripletas basura de ejemplo son: *(non - eat - vegetable)*, *(vegetation - eat - insect)* o *(toxic - property - dangerous)*, donde estas no guardan relación con el concepto principal. En este caso, el concepto principal o main concept elegido es "hamster", para ambos cuestionarios.

5.2.1. Cuestionario corto

La información de este cuestionario está sacada del recurso *TextStorm*. Las tripletas proporcionadas en este cuestionario son las que aparecen en la tabla 5.1.

Concepto sujeto	Relación	Concepto objeto
mouse	is a	pet
people	keep	pet
hamster	is a	crepuscular
hamster	feed on	seed
tail	property	short
pouch	property	long
hamster	have	tail
hamster	may look	slow
hamster	be	fast

Tabla 5.1: Tripletas disponibles para la encuesta corta

Ahora vamos a mostrar las respuestas. Tenemos que tener en cuenta que las respuestas son en inglés y están escritas por usuarios con diferentes niveles de dominio del idioma.

- *Hamster is crepuscular and may look slow, but it is fast. Hamster also has a short tail and feeds on seed. People keep pet, like mouses. Pouch is long.*
- *Hamsters are crepuscular and they feed on seeds. They have a short tail. Although they may look slow, they are fast.*

- *People tend to keep pets such as mice and in particular hamsters. They feed on seeds and their tail is a little bit short, whereas their pouches are very long. Hamsters may look slow but they are fast.*

- *A hamster has long pouches. It eat seeds. Hamsters may look slow. It has a short tail but it is fast. People keep pets, for example a mouse.*

5.2.2. Cuestionario largo

Este cuestionario lleva mucho más tiempo completarlo que el anterior, por ello se han obtenido menos respuestas. Las tripletas proporcionadas son las que aparecen en la tabla 5.2.

Concepto sujeto	Relación	Concepto objeto
hamster	is a	mammal
hamster	is a	rodent
hamster	is a	animal
hamster	capable of	spin on wheel
hamster	desires	food water and nice burrow
golden hamster	is a	hamster
hamster	at location	house
hamster	is a	small rodent that some person keep as pet
hamster	at location	hamster wheel
hamster	capable of	burrow
hamster	not desires	messy burrow
hamster	has a	cheek pouch
hamster	member of	cricetus
eurasian hamster	is a	hamster
hamster	is a	small rodent
hamster	not capable of	pilot airplane
hamster	is a	specie
hamster	related to	gerbil
hamster	related to	rat
hamster	related to	guinea pig
hamster	related to	mouse
hamster	is a	living thing
djungarian hamster	related to	hamster
gerbil	related to	hamster
hammy	related to	hamster
antihamster	etymologically derived from	hamster

Tabla 5.2: Tripletas disponibles para la encuesta larga

Las respuestas recibidas para este cuestionario son las que se enumeran debajo.

- *Hamster is a small rodent that some person keep as pet at house, a mammal animal member of cricetus. Hamster has a cheek pouch and it is capable od spin on wheel and burrow, but not capable of pilot an airplane. Hamsters desire food water and nice burrow, not necesary messy burrow. Some types of hamters are golden hamster, eurasian hamster or djungarian hamster. Hamster is related to gerbil, rat, guinea pig, mouse and hammy.*
- *A hamster is a small rodent, a mammal and an animal. It is also a specie and a live thing. It is a member of cricetus. You can find a hamster at a house or at a hamster wheel. Hamsters can spin on wheels and burrow, but hamsters can't pilot airplanes. They desire food, water and nice burrow, but they hate messy burrows. Hamsters has cheek pouch. Golden hamsters and Eurasian hamsters are kinds of hamster.*

5.2.3. Conclusiones del experimento

Ahora vamos a analizar las respuestas de los usuarios para sacar los aspectos característicos de sus textos y poder hacer una plantilla para la generación de lenguaje natural. Como en la segunda encuesta que hemos explicado, el cuestionario largo (subsección 5.2.2) no hemos obtenido suficientes respuestas como para poder valorar los aspectos de los textos, no podremos estudiar los textos cortos y largos por separado. Pero tendremos en cuenta las respuestas de este cuestionario para las observaciones que hagamos en los cortos.

Solo algunos usuarios que rellenaron las encuestas utilizaron el signo de puntuación punto y aparte para separar ciertas secciones. Tampoco todos los usuarios incluyen todos los datos proporcionados (tanto si es información que no se puede relacionar como si son datos útiles), aunque como ya hemos mencionado anteriormente, en este trabajo no vamos a preocuparnos por cubrir este aspecto (*determinación de contenido*). En lo que sí están de acuerdo es en colocar primero las relaciones que indican cualidades y atributos, es decir las relaciones Es (Is) Es_Un (Is_A), Comer (Eat) y las relaciones Tiene (Has). Los usuarios tienden a ordenar los conceptos objeto de las relaciones Es (Is) de manera que primero aparecen los más concretos y luego los más generales en la enumeración (como puede verse con *rodent* $<$*mammal* $<$*animal*). Para los conceptos que no pueden establecer esta escala con otros, el orden no está claro y pueden aparecer en cualquier lugar.

Los usuarios también tienden a utilizar frases con contraste y frases usando agregación con aquellas relaciones que lo permiten.

El estilo que utilizan los usuarios es variado, algunos escriben de forma más simple y compacta y otros utilizan vocabulario y estructuras más variadas. En los textos con más información y más relaciones, los usuarios añaden después de las relaciones mencionadas en el párrafo anterior otras relaciones como Se_Encuentra_En (At_Location), Es_Capaz_De (Capable_Of) o Desea (Desire). Al final del texto, añaden ejemplos del concepto, en nuestro caso esta relación esta representada por Está_Relacionado_Con (Related_To).

Algunos de los aspectos que hemos mencionado resultan interesantes para nuestro cometido. El uso de contraste entre conceptos es frecuente en los textos de los usuarios. Pero para nosotros esto supone una dificultad, ya que tenemos que ser capaces de reconocer la contraposición de dos relaciones dadas para la información que comparte el concepto sujeto. Como existen varios casos en los que parece no haber un criterio claro al organizar la información, podemos utilizar como herramienta el orden de estas relaciones para dotar de aspectos diferentes a los textos. Esto se conseguiría añadiendo un factor de aleatoriedad a la planificación del texto para estas relaciones.

5.3. Generación elaborada de texto

Fijándonos en las conclusiones del apartado 5.2.3 anterior que hemos podido obtener de los textos generados por los usuarios, podemos realizar una plantilla (método también conocido como *canned text*, el cual ya mencionamos en el apartado 2.1 de este trabajo) con la que poder generar texto de forma similar. Podremos observar que las conclusiones con las que se va a generar la plantilla se basan principalmente en secuenciar las relaciones como los usuarios lo han hecho. Veremos que estas relaciones son fijas y pueden no estar presentes en algunos grafos para los que se generen las descripciones, por tanto esta plantilla puede considerarse específica.

Primero localizamos las relaciones que tienen como concepto sujeto el concepto principal del que se va a hablar. A continuación, seleccionamos las relaciones Es_Un (IS_A), que nos van a permitir realizar una descripción del concepto del que hablamos. En este momento podríamos ordenar los conceptos objeto para que apareciesen en un cierto orden, como hemos dicho en las conclusiones 5.2.3, de los más concretos a los menos concretos. Como en este momento no disponemos de mecanismos para conocer qué conceptos guardan alguna relación con otros, dejaremos este aspecto para más adelan-

te. También podríamos aplicar una reducción a este conjunto para eliminar datos y resumir, pero en este trabajo no vamos a tratar la fase de determinación de contenido, suponemos que ya se ha realizado esta tarea y nos viene dada como entrada.

Como no podemos averiguar aún si un concepto es contable o incontable, usaremos los resultados de las encuestas, que se realizaron para conceptos contables. La plantilla coge el concepto sujeto de la tripleta y añade una terminación para hacerlo plural, luego genera el plural de la tercera persona del verbo correspondiente, en este caso se trata del verbo Ser (Be), por tanto se generaría "son" ("are"). Para seguir escribimos el primer concepto objeto en su forma plural (añadiendo una "s"). Ahora cambiamos la primera letra para que sea mayúscula y terminamos de añadir todos los conceptos objeto restantes de la relación teniendo en cuenta que en la enumeración el último concepto debe estar precedido de un conector aditivo. Finalmente se añade un punto como signo de puntuación para terminar la frase.

Un posible ejemplo de salida de texto hasta este punto sería *"Squirrels are mammals, animals and rodents."*.

La siguiente frase se generará con las tripletas que tengan como relación entre los conceptos el verbo Comer (Eat). El proceso de generación del texto es similar al contado anteriormente. En este caso cambiaremos el concepto sujeto del que hablamos y pondremos en su lugar el pronombre personal correspondiente, tercera persona del plural. Además modificaremos los conceptos objeto para que aparezcan el plural y la frase sea correcta. Si se ha podido generar texto en estas dos fases anteriores, ambas partes quedarán en un mismo párrafo. Por el contrario se generará un párrafo nuevo.

La frase *"They eat nuts, seeds and fruits."* es un ejemplo de frase que esta parte de la funcionalidad generaría.

La última de las relaciones que siempre intentan incluir primero los usuarios es la relación Tiene (Has). Con lo cual la añadiremos a continuación. En este tipo de relaciones nos es más sencillo encontrar en los datos de entrada tripletas que pueden encadenarse para hacer más precisa la información del texto. Las tripletas *(squirrel - have - tail)*, *(tail - property - haired)* y *(tail - property - smooth)* darían lugar a una frase como esta: *"Squirrels have a smooth haired tail."*. Para conseguir producir este texto se ha decidido que se generará una frase para cada relación hasta que se encuentre, separándolas por el signo de puntuación punto y seguido. Aunque podríamos generar una enumeración con los conceptos y adjetivos de estos, sin embargo nos arriesgamos a que haya muchos adjetivos y/o muchos conceptos y dicha enumeración tenga demasiada longitud, haciéndose difícil de entender. Para esta parte de la generación también se seguirán las reglas correspondientes

para poner mayúsculas y escribiremos el concepto sujeto en cada oración. Entonces se recorrerán todas las relaciones Tiene (Has) y para cada concepto objeto obtenido se realizará una búsqueda para encontrar las relaciones Propiedad (Property) de esta para concatenarlas y situarlas como adjetivos en la frase.

La siguiente relación en la que nos vamos a fijar es Se_Encuentra_En (At_Location). Para que el texto suene más natural, en vez de seguir el mismo patrón que en las anteriores relaciones, en este caso utilizaremos una estructura diferente. El resultado que queremos producir es una frase como *"You can find squirrels in trees."*.

En cualquiera de los anteriores casos, si no se encuentran tripletas con la relación que buscamos, se omitirá la generación del texto correspondiente y la frase en cuestión no se generará y no se añadirá al resto del texto que ya se tiene.

Como último paso se seleccionan el resto de tripletas y se generan las frases correspondientes. Únicamente las tripletas que contengan en su parte izquierda o parte derecha el concepto principal del que se habla (main entity) generarán texto. Para aquellas que tienen el concepto en la parte izquierda, de forma aleatoria, utilizaremos el propio concepto o su pronombre personal correspondiente para generar la frase. El resto de la oración se genera añadiendo la relación y después construimos una enumeración con los conceptos objeto. Para las tripletas que contienen el concepto principal en la parte derecha y además tengan como relación Relacionado_Con (Related_To), almacenaremos el concepto sujeto. Cuando terminemos de recorrer todas las tripletas generaremos una frase que mencione estos conceptos diciendo que son tipos de la entidad de la que hablamos. "Cat squirrels, fox squirrels and asiatic flying squirrels are kinds of squirrels." es un ejemplo para ver cómo queda un fragmento de texto de este tipo.

Después de añadir, si corresponde, los ejemplos del concepto principal, se concatenan las frases generadas por el resto de tripletas.

El siguiente texto representa un ejemplo completo de salida del sistema conseguido mediante la ejecución del módulo de generación de textos elaborados, que es el que hemos descrito en este apartado. Hemos elegido el concepto "hamster" para poder ver la diferencia con el generador anterior y para poder comparar la salida con los textos de los usuarios.

Hamsters are mammals, rodents, animals, small rodent that some person keep as pets, small rodents, bands, species and live things. You can find hamsters in houses and in hamster wheels. Djungarian hamsters, gerbils and hammys are kinds of hamsters. Hamster related to gerbil, rat, guinea pig

and mouse. Hamster capable of spin on wheel, burrow and from flimsy cage. They desires food water and nice burrow. Hamster not desires messy burrow. They has a cheek pouch. They member of cricetus. They not capable of pilot airplane.

Este generador de texto resulta algo más complejo que el anterior que hemos visto en la sección 5.1. La parte del estudio es una parte que no podemos controlar directamente, ya que depende de que los usuarios realicen los cuestionarios. Tampoco resulta trivial encontrar los patrones comunes de las respuestas de los usuarios, ni las diferencias que aportan creatividad, ya que hay que corregir los textos porque se ha incluido información que no se les ha proporcionado o las oraciones no están construidas correctamente. Sin embargo, hemos conseguido identificar el orden en el que suelen mencionar las relaciones, y el uso de contraste y enumeraciones.

Dependiendo de la implementación del código podemos conseguir frases más complejas y correctas, pero en nuestro caso lo hemos querido mantener lo más simple posible. Esta funcionalidad también resulta sencilla de desarrollar en un único módulo, así que no es necesario crear nada adicional para ejecutarlo.

La parte de extensibilidad de este código no es tan buena como se desearía. Para añadir funcionalidad extra o modificar el comportamiento para mejorarlo, por ejemplo utilizando contraste entre relaciones, sería necesario replantear el algoritmo que se encarga de la generación. Esto es debido a que en el caso que hemos mencionado, el contraste requiere extraer dos o más tripletas que se pueden encontrar agrupadas con otras, lo que rompe el orden por el que se ordena la información y no está claro cuál debería ser el sitio coherente para introducir la frase con el contraste.

Por otro lado, la calidad del texto es mucho mejor que la conseguida hasta ahora. Esta calidad se ha conseguido para el tipo de textos que se ha estudiado, y puede servir para otros. Hasta ahora los datos que se nos han proporcionado y el objetivo de este trabajo se centra en textos descriptivos que permitan explicar conceptos. Pero no se puede asegurar que se mantenga si el grafo semántico no encaja bien la generación de un texto descriptivo. Esto se fundamenta en que las relaciones que hemos utilizado para la generación son concretas, como Comer (Eat) o Tiene (Have), relaciones que puede que otros grafos no contengan, por ejemplo aquellos relacionados con eventos.

5.4. Generación de texto con planificación

En los anteriores generadores que hemos desarrollado no hemos tenido nada más que una etapa, es decir, con un único módulo era suficiente para tener la funcionalidad deseada con la modularidad esperada. Ahora vamos a desarrollar un tipo de generación más compleja, esta tendrá una etapa de planificación y otra etapa de realización o escritura (conversión a texto). Esto necesitará que dichos módulos se ejecuten en cierto orden y de forma secuencial.

Para realizar una generación de texto con planificación es necesario crear un módulo que se encargue de ello, y a continuación pasarle la información a otro que se encargue de generar el texto tal y como se ha planificado. Por esta razón, en el módulo de generación de texto que hemos visto en la figura 3.4, en el capítulo 3.6, vamos a introducir un pipeline y a ejecutarlo, en lugar de usar un solo módulo. Crear un pipeline rompe la idea que habíamos mantenido hasta ahora de que en cada fase del sistema pudiéramos ejecutar los módulos que queramos y en el orden que deseemos. En contraposición a esta ventaja hay que remarcar que conseguiremos una mejor estructura en la aplicación y esta nos dará modularidad y extensibilidad para incorporar mejoras en el futuro. En cualquier caso, si resulta molesto tener que ejecutar una serie de módulos en vez de uno, se puede encapsular la ejecución y toda la funcionalidad en un módulo a modo de caja negra.

5.4.1. Planificador simple

El pipeline que tenemos consta de un planificador y un realizador de texto. El planificador va a recibir la estructura de datos con las tripletas y el concepto principal del que tratan los datos. Después lo primero que se crea es la siguiente estructura de datos que conformará la salida de este módulo, y por tanto la entrada del realizador de texto. Esta estructura necesita tener orden, ya que en un texto el orden en el que se cuenta la información importa e influye en el lector y su comprensión, así se puede planificar la aparición de la información. Por tanto, utilizaremos una lista de instrucciones (*Instruction*). Estas instrucciones tienen en común un método que permite solicitar el resultado, por lo que tendremos una herencia de clases (podemos consultarlo en la figura 5.1), pero difieren en el constructor, porque los diferentes tipos de palabras que vamos a necesitar tienen diferentes formas de configurarse. A continuación vamos a enumerar y a explicar brevemente cada una.

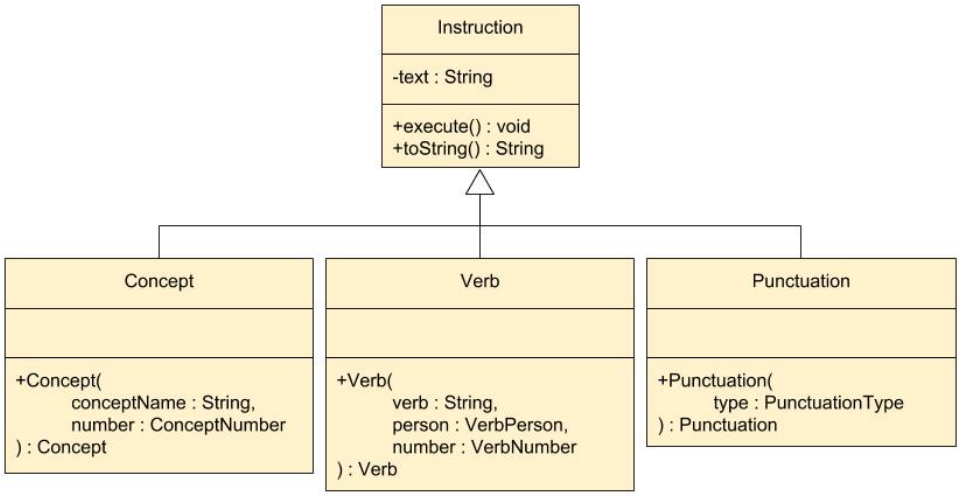


Figura 5.1: Diagrama de instrucciones

- *Concept*: La forma de construir una instrucción de este tipo es proporcionando el concepto o sustantivo que se desea y el número, que podrá ser plural o singular. Es la forma de crear sustantivos o conceptos.
- *Verb*: En este caso necesitaremos la forma base (infinitivo) del verbo, la persona y el número, así podremos conjugar los verbos.
- *Puntuation*: Podemos elegir signos de puntuación y otros elementos necesarios para la construcción del texto, por ahora se han incluido el punto, la coma, el espacio, el salto de linea y los símbolos de interrogación y exclamación (únicamente los de cierre, ya que los textos generados serán exclusivamente en inglés). Para utilizar otros extra resulta trivial añadirlos, se realiza mediante un enumerado.

Como se trata de un planificador, lo primero que vamos a hacer es ejecutar el módulo de secuenciación de relaciones (4.4) a modo de utilidad, con esto conseguimos asegurarnos que los datos que se encuentran al principio son los que creemos más importantes. El resto de módulos de utilidades (podemos consultarlo en el capítulo 4), no los encontramos necesarios, así que no ejecutaremos ninguno de ellos.

Para realizar la planificación se realiza un bucle que recorre todo el grafo semántico, es decir, todas las tripletas de las que disponemos. Se añaden las instrucciones para indicar el concepto sujeto en plural, el verbo conjugado a la tercera persona del plural y para finalizar se recorren los conceptos objeto para escribir una enumeración, teniendo en cuenta que es necesario utilizar comas y entre el último y el penúltimo concepto un conector aditivo. Al final

de cada frase se añade la puntuación adecuada, que es un punto y un salto de línea.

Para terminar de entender el resultado de esta fase vamos a partir de un grafo semántico muy pequeño de ejemplo y a crear en la tabla 5.3 las instrucciones ordenadas que se generarían según lo anteriormente explicado.

Grafo semántico de entrada para el ejemplo:

(hamster - Be - mammal) (hamster - Be - rodent) (hamster - Be - animal)

Representación interna del sistema para dicho grafo semántico proporcionado:

(hamster - Be - [mammal, rodent, animal])

Concept("hamster", ConceptNumber.PLURAL)
Punctuation(PunctuationType.BLANK)
Verb("Be", VerbPerson.third, VerbNumber.plural)
Punctuation(PunctuationType.BLANK)
Concept("mammal", ConceptNumber.PLURAL)
Punctuation(PunctuationType.COMMA)
Concept("rodent", ConceptNumber.PLURAL)
Concept(" and ", ConceptNumber.SINGULAR)
Concept("animal", ConceptNumber.PLURAL)
Punctuation(PunctuationType.STOP)
Punctuation(PunctuationType.NEXT_LINE)

Tabla 5.3: Instrucciones generadas para la generación de texto con planificación simple

5.4.2. Realizador de texto

El realizador de texto (que también podemos llamar generador de texto, aunque pueda resultar confuso con los otros módulos de generación de texto), como hemos visto antes, tiene como entrada la lista de instrucciones generada por el planificador. El primer paso es crear el campo que se va a devolver, y también se crea e inicializa a verdadero (true) una variable (que usaremos como flag) que nos indicará si la próxima letra a escribir es mayúscula o no. Después, secuencialmente recorre de principio a fin dicha lista de instrucciones concatenando los resultados de las sus ejecuciones y teniendo en cuenta la variable de las mayúsculas, tanto para cambiar el carácter correspondiente como para reconocer cuándo se escribe un punto, y por tanto se debe volver a activar la variable.

Para conseguir conjugar el verbo se hace uso de un recurso llamado *Ultralingua*[1]. En caso de no poder encontrar el verbo que se busca, este recurso muestra información de error, y al detectarla nosotros, hacemos que se devuelva la forma en infinitivo, que es de la que partíamos. Por ahora solo solicitamos los datos en presente, ya que es la única forma que vamos a utilizar, pero el recurso que se ha desarrollado está preparado para soportarlo.

Vamos a mostrar un par de ejemplos de salida para ver el resultado de este generador de lenguaje natural. Los datos de entrada para el primero son los que ya hemos mencionado en el apartado anterior (5.4.1), podemos consultar las instrucciones que llegan como entrada en la tabla 5.3. De esta forma podemos ver un ejemplo de todos los datos intermedios que se generan y de la entrada y la salida.

Primer ejemplo:

Hamsters are mammals, rodents and animals.

Datos del segundo ejemplo:

(hamster - be - [mammal, rodent, animal,
small rodent that some person keep as pet, small rodent, band, specie, live thing])
(hamster - related to - [gerbil, rat, guinea pig, mouse])
(hamster - capable of - [spin on wheel, burrow, from flimsy cage])
(hamster - at location - [house, hamster wheel])
(hamster - desires - [food water and nice burrow])
(hamster - not desires - [messy burrow])
(hamster - has a - [cheek pouch])
(hamster - member of - [cricetus])
(hamster - not capable of - [pilot airplane])
(golden hamster - be - [hamster])
(eurasian hamster - be - [hamster])
(hamster cheek - be - [hamster])
(djungarian hamster - related to - [hamster])
(gerbil - related to - [hamster])
(hammy - related to - [hamster])
(antihamster - etymologically derived from - [hamster])

Segundo ejemplo:

Hamsters are mammals, rodents, animals, small rodent that some person keep as pets, small rodents, bands, species and live things. Hamsters related to gerbils, rats, guinea pigs and mouses. Hamsters capable of spin on wheels,

[1] `http://api.ultralingua.com/`

burrows and from flimsy cages. Hamsters at location hamster wheels. Hamsters desire food water and nice burrows. Hamsters not desires messy burrows. Hamsters has a cheek pouchs. Hamsters member of cricetuss. Hamsters not capable of pilot airplanes. Golden hamsters are hamsters. Eurasian hamsters are hamsters. Hamster cheeks are hamsters. Djungarian hamsters related to hamsters. Gerbils related to hamsters. Hammys related to hamsters. Anti-hamsters etymologically derived from hamsters.

El generador que hemos desarrollado y hemos explicado en este apartado presenta más complejidad que los dos anteriores, tanto en el aspecto de programación como en el aspecto de arquitectura. Por la parte de programación ha hecho falta crear las instrucciones que se encargan de expresar recoger la configuración que va a ser necesaria para la creación del texto. También se necesitan recursos adicionales para poder conseguir la funcionalidad deseada y transformar las instrucciones en las palabras que necesitamos. Tampoco hay que olvidar que se necesita el realizador superficial para poder construir el texto final de salida. Por el aspecto de la arquitectura tenemos que mencionar que es necesario crear el pipeline, en vez de usar un único módulo, que secuencia las funcionalidades de los módulos y permite más flexibilidad a la hora de introducir más módulos de la generación de lenguaje natural.

Gracias a lo anteriormente mencionado, el aspecto del pipeline a base de módulos, disponemos de una arquitectura en la que podemos añadir o cambiar los módulos por otros de una forma muy sencilla. Las estructuras que utilizamos (listas de instrucciones con herencia) y la forma de trabajar con ellas permiten que el código sea más manejable a la hora de aplicar cambios o añadir nuevas clases, es decir, tenemos un código más extensible.

La calidad del texto también ha mejorado, aunque el aspecto del texto es similar al generado por el generador básico (descrito en el apartado 5.1), podemos notar que gracias a la funcionalidad añadida al ejecutar las instrucciones los verbos se conjugan y se hace un mejor uso de las formas en plural y singular.

5.5. Generación de texto simplificada

Se ha observado que ciertas tripletas de los datos originales contienen relaciones que otras tripletas ya cubren, por ejemplo las tripletas *(hamster - Is_A - mammal)* y *(hamster - Is_A - rodent)*. En este ejemplo el concepto "mammal" está contenido en el concepto "rodent", el cual además amplía la información. Por esta razón podemos elegir crear un texto con el contenido de ambas tripletas, repitiendo la información, o podemos optar por generar

un texto que solo incluya el concepto que más contenido abarque, sin perder información. Esto es posible gracias a que en la fase previa de determinación del contenido se elige la información que se tiene que contar en el texto, y en nuestro caso seguimos respetando la decisión que en esta fase se tomó.

Para detectar estas coincidencias en el significado de los conceptos, vamos a utilizar la herramienta *WordNet*, que ya analizamos y explicamos en la sección 2.6.1. Utilizaremos esta herramienta para buscar los *direct hypernym* de un concepto objeto y a los resultados se les aplicará el mismo proceso. Repitiendo este tratamiento, puede que en algún punto, uno de estos resultados coincida con otro de los conceptos objeto (de la lista original) de los que disponemos. Solo buscamos y comparamos los conceptos objeto de cada tripleta, así ambos comparten el concepto sujeto y la relación, por lo que encontraríamos lo que estábamos buscando, un concepto con su significado contenido en otro. Un ejemplo de este caso sería el siguiente:

Para el concepto "hamster", que es reconocido por el sistema como la entidad principal (main entity) sobre la que tratan los datos, podemos encontrar las siguientes tripletas (extraídas del recurso *ConceptNet 5*):

(hamster - Is_ a - mammal)
(hamster - Is_ a - animal)

Si buscamos los *direct hypernym* de "mammal" encontramos las relaciones que podemos ver en la figura 5.2.

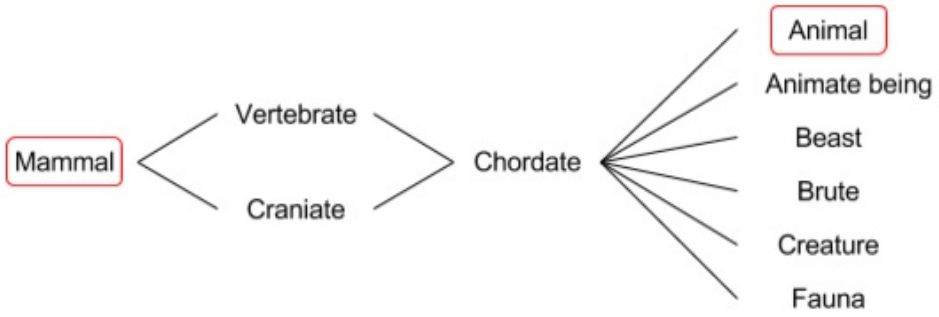


Figura 5.2: Gráfico de hiperonimia de ejemplo

Como podemos observar en el diagrama 5.2, hemos encontrado las relaciones entre dos tripletas, con esto, el sistema conoce mejor las relaciones y es capaz de generar un mejor texto. El sistema se basa en que sabe que la palabra "mammal" es un concepto más específico (que posee todos los rasgos semánticos, o semas, del otro concepto) de la palabra "animal". Por ello se puede construir la siguiente frase, utilizando el concepto más específico como adjetivo y manteniendo el más general como sustantivo en la frase:

A hamster is a mammalian animal.

Esta funcionalidad no se ha implementado en el sistema aún, ya que no se encuentra necesaria y conserva la redundancia y la longitud del texto. Además es necesaria una herramienta para convertir el sustantivo en un adjetivo, en nuestro caso podríamos utilizar *WordNet* para conseguirlo. Sin embargo, la funcionalidad que explicamos a continuación sí está implementada y hace que la frase resultante no sea redundante y se acorte su longitud.

En caso de querer resumir o no utilizar la palabra "animal" podríamos omitirla, ya que "mammal" ya contiene su significado sin perder información semántica. Con lo que generaríamos la frase:

A hamster is a mammal.

La cantidad de búsquedas que realizamos se puede cambiar, pero por defecto la hemos fijado en veinticinco, realizando pruebas parece no consumir tiempo excesivo en las consultas y permite un margen amplio para encontrar coincidencias. Se debe tener en cuenta que la distancia a la que se encuentran los conceptos que busquemos dependerá directamente de ellos mismos, por lo que no se puede calcular exactamente su parecido de antemano para encontrar las coincidencias.

En el caso en el que se encuentren coincidencias, lo que hacemos es borrar el concepto de la lista original que ha coincidido con uno de los hiperónimos, obteniendo finalmente todas las tripletas sin redundancias.

Para finalizar y generar las instrucciones que darán lugar al texto planificado se hace uso del planificador simple que ya tenemos y hemos explicado en el capítulo 5.4.1.

Este generador es muy similar al *planificador simple*, ya que lo utiliza como base para generar el texto. Lo único que se añade en este generador es la funcionalidad que permite detectar los contenidos semánticos que están incluidos en otros conceptos, y decidir si omitir o no el concepto redundante. Debido a esto, la complejidad del código que se necesita programar y de la arquitectura necesaria para ejecutarlo prácticamente se mantienen igual. Lo mismo pasa con el aspecto de extensibilidad. Por otro lado, la calidad del texto final aumenta tanto si se decide omitir, haciendo que el texto resultante sea más corto y preciso, como si se decide añadir como adjetivo.

Aunque la salida producida por este generador es muy parecida al anterior, vamos a mostrar un ejemplo, eligiendo como entidad principal la misma que en el *planificador simple*, así podremos ver las diferencias entre ambos.

Ejemplo:

Hamsters are rodents, small rodent that some person keep as pets, small rodents, bands, species and live things. Hamsters related to gerbils, rats, guinea pigs and mouses. Hamsters capable of spin on wheels, burrows and from flimsy cages. Hamsters at location hamster wheels. Hamsters desire food water and nice burrows. Hamsters not desires messy burrows. Hamsters has a cheek pouchs. Hamsters member of cricetuss. Hamsters not capable of pilot airplanes. Golden hamsters are hamsters. Eurasian hamsters are hamsters. Hamster cheeks are hamsters. Djungarian hamsters related to hamsters. Gerbils related to hamsters. Hammys related to hamsters. Antihamsters etymologically derived from hamsters.

5.6. Comparación de los generadores

Al final de cada sección de este capítulo en la que se explica un generador de texto, hay una valoración y crítica de las características del mismo. Dichos generadores se desarrollaron secuencialmente a medida que el proyecto avanzaba, por tanto es normal que cada vez que se crea uno nuevo tenga mejor valoración, en los aspectos más mejorables de su predecesor. Ahora vamos a situarnos con perspectiva y a decidir cuál es el mejor candidato y qué características son más remarcables de cada uno.

El generador básico es un generador muy sencillo de programar y de integrar, por tanto es el recomendado si se desea generar texto sin perder tiempo en la implementación. Como desventajas hay que remarcar que es muy poco extensible y la calidad del texto generado es baja en comparación con los otros.

El generador de texto elaborado, presenta una principal diferencia con respecto al resto y es que requiere un estudio de los textos de los usuarios, con todo lo que ello conlleva. El tiempo empleado en realizar dicho estudio, sumado a la programación de las plantillas, es muy alto. Además, estas plantillas podrían tener que cambiar a medida que se consiguen estudiar más textos y de diferentes características, lo que suma un extra al tiempo empleado. Este método no es característico por su complejidad de programación y tiene una arquitectura que consta de un solo módulo, por lo que resulta muy sencillo de integrar y utilizar. No tiene muy buena extensibilidad, esto es debido a lo mencionado anteriormente, es dependiente del estudio y el avance de las plantillas. Sin embargo, este generador da muy buenos resultados en cuanto a la calidad de los textos generados. Resulta una muy buena opción si se desea generar textos característicos con buena calidad.

El generador con planificación simple presenta una calidad mejor que el generador básico, pero no suena tan natural como el generador de texto elaborado. Esta carencia se suple con una mejora muy considerable en la extensibilidad, que lo que permite es mejorar con el tiempo de forma más sencilla dicha calidad. Aunque la complejidad de la arquitectura necesaria para crearlo es mayor, el aspecto de programación no se ve afectado. Un sistema con estas características resulta ideal para comenzar un proyecto de cierta duración, debido a que es razonablemente sencillo y fácil de mejorar.

El generador con planificación simplificada, como ya sabemos, es una versión del ya mencionado generador con planificación simple. Por ello conserva sus características y solo mejora la calidad del texto producido. Es el siguiente paso para aquellas situaciones donde elegiríamos en primer lugar un generador con planificación.

Para que quede más claro, se representa de forma gráfica en la tabla 5.4 las valoraciones de cada aspecto que hemos tenido en cuenta. Dichas valoraciones se representan de cero a cinco estrellas, donde cero es la peor nota y cinco es la mejor.

Vamos a adjudicar diferentes pesos a las diferentes características de la aplicación (dificultad de programación, complejidad de la arquitectura, extensibilidad del código y calidad del texto final). El aspecto más importante sin duda para este trabajo es la calidad final del texto que generamos como salida. En el siguiente puesto, valoraremos mejor la extensibilidad del sistema. Nos dimos cuenta de la importancia de esta característica tras desarrollar el *generador elaborado* (apartado 5.2), y corroboramos que estábamos en lo cierto tras ver la facilidad con la que añadimos funcionalidad en el *generador simplificado* (apartado 5.5). El siguiente aspecto mejor valorado es la dificultad de programación, que nos indica el tiempo necesario de implementación (dependiendo de la habilidad del programador). Por último, calificaremos la arquitectura como la característica menos importante, esto es porque el esfuerzo empleado en crearla solo se hace una vez y, en principio, no se modifica. En resumen, multiplicaremos la puntuación del apartado calidad por cuatro ya que es el aspecto más importante para nosotros, el aspecto de extensibilidad verá aumentado tres veces su valor por ser el siguiente. Los restantes aspectos, programación y arquitectura, se multiplicarán por dos y por uno su valor respectivamente por ser los dos aspectos menos valorados.

La valoración de las características se representará en estrellas. Cuantas más estrellas mejor se considera ese aspecto del generador.

La dificultad de programación se medirá como el tiempo empleado en desarrollar el código, 5 horas como mucho equivale a cinco estrellas, emplear

de 5 a 10 horas cuatro, de 10 a 15 tres, de 15 a 20 dos y utilizar más de 20 horas una estrella.

La arquitectura se valorará según la cantidad de módulos que tenga la parte de generación del texto, una arquitectura con un módulo obtendrá la máxima puntuación, cinco estrellas. Por cada módulo adicional se restará una estrella, hasta un mínimo de una. Hay que tener en cuenta que estamos considerando módulos los que tenemos implementados en el sistema, aunque estos engloben varios del esquema de un sistema de generación de lenguaje natural. Estos submódulos solo serán contados si existe una entidad en el código adicional que se llame desde el módulo superior.

La extensibilidad del código es la facilidad con la que se pueden aplicar cambios al sistema, generalmente ampliando su funcionalidad. Por esta razón valoraremos este apartado exactamente igual que el apartado de la dificultad de programación. Estimaremos la dificultad de programación de añadir el módulo de microplanificación (microplanning) al sistema. Este módulo tendrá una complejidad similar a la del primer módulo del *planificador simple*.

La calidad del texto muestra lo que es capaz de conseguir la aplicación. La mejor forma de la que se podría medir dicha calidad es generando diferentes textos sobre conceptos variados, que presenten diferentes características, y generar una encuesta para que se evalúe la calidad de estos por personas. Debido a que no se dispone de tiempo suficiente para esperar las respuestas por parte de los usuarios de la encuesta, se ha decidido valorar la calidad de una forma alternativa. En este método de evaluación se considera que las descripciones deben mostrar las características esperadas de un texto, así que sumaremos valor por cada buena práctica que este presente. Estas cualidades en la que vamos a fijarnos son, por ejemplo, si las palabras bien escritas y concuerdan en género y número según corresponda, la organización de ideas en una misma oración o párrafo y uso de conectores aditivos entre estas, la correcta conjugación o transformación del verbo que está representado como relación, que presente variedad en el vocabulario para los conceptos que aparecen varias veces, la redundancia entre las ideas que se mencionan en el texto o contraste de las relaciones y conceptos que lo permitan. Por cada una de estas propiedades que presente el texto el generador ganará una estrella en su valoración.

Finalmente, vamos a elegir el mejor candidato. No solo por sumar más puntuación en la valoración que se encuentra en la tabla 5.4, si no también porque resulta tener más nivelados todos los aspectos. Sobre todo, es importante elegir un generador por se extensibilidad, ya que es la cualidad que va a permitir mejorar de forma más rápida y sencilla la calidad, que es en

Generador	Programación	Arquitectura	Extensibilidad	Calidad	Total
Básico	*****	*****	*	**	26
Elaborado	***	*****	**	****	33
Planificación simple	***	****	*****	***	37
Simplificado	***	****	*****	****	41

Tabla 5.4: Valoraciones de los generadores

definitiva el objetivo de este trabajo. El aspecto de la programación siempre va a disminuir cuanto más tiempo se le dedique a mejorar la funcionalidad. La arquitectura dependerá del planteamiento inicial sobre el que se apoya la aplicación y que define la forma de trabajar, esta es una cualidad que se mantiene en el tiempo, a menos que esta permita modificarse y se considera mejor cuanto más simple. La calidad es una característica muy variable, añadiendo funcionalidad se puede conseguir disminuir o aumentar, y en muchas ocasiones resulta una medida subjetiva. Por todo ello, el generador elegido es el planificador simplificado. La opción óptima es poder conservar y avanzar en los estudios de las plantillas del generador de texto elaborado para ver las características que hacen que el texto suene mejor y más natural y aplicar estos conocimientos en el planificador simplificado, que resulta más sencillo de mejorar.

Capítulo 6

Conclusiones

En la sección 1.2 propusimos una serie de objetivos para este trabajo y ha llegado el momento de valorar si estos se han conseguido o no. El primero de nuestros objetivos era localizar y estudiar los recursos disponibles y seleccionar aquellos fáciles de utilizar y que contengan datos fiables. En el apartado 2.6 queda reflejado este trabajo, por tanto podemos dar como cumplido y terminado este objetivo. Nuestro siguiente objetivo requería adaptar cada uno de los datos de estos recursos a un formato interno del sistema para que este fuera independiente de la entrada. Además, en este objetivo también se propone facilitar en la medida de lo posible la generación del texto. El capítulo 4 está dedicado exclusivamente a el preprocesamiento de la entrada antes de la generación de lenguaje natural. Consideramos que con toda la funcionalidad que está descrita, podemos convertir los recursos a un formato interno y además proporciona traducciones de las relaciones para hacer que el grafo sea más conexo y conocer mejor los datos. Por las razones que acabamos de exponer damos por cumplido y satisfecho el segundo objetivo que propusimos. Como tercer objetivo, tenemos la creación de un generador de descripciones que sea capaz de presentar las características de los conceptos de entrada del sistema. En el capítulo 5 podemos encontrar los generadores que hemos ido implementando y mejorando, consideramos que varios de ellos producen textos con la calidad suficiente como para dar este objetivo como cumplido. Por último, nos propusimos el objetivo de introducir la aplicación resultante de este trabajo en la plataforma del proyecto *ConCreTe*. Por falta de tiempo y añadiendo que no disponemos por ahora de la herramienta generada en el trabajo fin de máster mencionado en la sección 2.5.2, la cual produce como salida la entrada que nuestro sistema espera, no hemos introducido el servicio web en el proyecto. Este último objetivo no podemos darlo aún por cumplido.

Ahora vamos a mencionar otras observaciones de las que nos hemos dado cuenta a lo largo del desarrollo del proyecto.

La generación de textos no es una tarea trivial de la informática. Sobre todo, cuando queremos darle una naturalidad propia de las personas y simular su comportamiento. Para ello hemos visto que hace falta aplicar varios campos, como puede ser la creatividad computacional o la inteligencia artificial, para crear los textos. También se necesita consultar los avances que se han hecho para conseguir este objetivo y de esta forma poder valorar cómo abordar el problema que se nos presenta, así como conocer los recursos disponibles que nos pueden ser de gran ayuda a la hora de conseguir datos y funcionalidad útil.

Este trabajo al formar parte de un proyecto mayor, *ConCreTe*, del que hemos hablado en la sección 2.5, se tiene que adecuar a su forma de trabajo y a sus necesidades a la vez que se investiga para conseguir las metas que se propone. Como efecto colateral de la colaboración entre los llamados *partners*, el trabajo se ve obligado en parte a trabajar con las cosas desarrolladas por estos, lo que lleva a que realicemos una función de tester y sugiramos mejoras que nos ayuden a ambos.

El *textifier* es una herramienta, de la que hablamos en el capítulo 3, que se desarrollo en este trabajo como primera aproximación a la generación de texto para los datos de entrada que se tenían en el estado en el que se encontraba el proyecto en ese momento. Para ojos de un lector, el texto resultante de la ejecución del sistema tiene buena calidad y suena bastante natural, esto es porque los datos que se manejaban son fragmentos de frases escritas por personas, a mano. Pero a ojos del programador o del investigador no resulta tan interesante trabajar con esos datos, ya que te proporciona poca flexibilidad a la hora de crear los textos, lo que reduce en gran medida la característica de originalidad y creatividad del sistema. Las limitaciones detectadas después de probar el sistema en conjunto revelaron que es mejor punto de partida utilizar datos conceptuales mucho más simples, lo que da lugar al siguiente objetivo, crear desde cero un generador de texto para los nuevos datos de entrada. Con todo esto en mente, un mejor planteamiento desde el principio habría supuesto un avance más rápido del proyecto.

El desarrollo iterativo que se ha seguido para la creación de los diferentes generadores de lenguaje natural ha proporcionado varias herramientas para conseguir diferentes textos. Así se han realizado varias propuestas que van mejorando las propiedades en las que menos destacaban las anteriores. Estos módulos tienen características diferentes y han permitido ver algunas de las cualidades de los textos generados por personas, lo que resulta útil y muy interesante para hacer que la salida del sistema se parezca más a la

que podría generar un humano. De la misma manera, hemos podido darnos cuenta de que la arquitectura del sistema puede complicarse para mejorar la extensibilidad del código, facilitando el trabajo futuro, las posibilidades del generador y la calidad del texto de salida.

Chapter 6

Conclusions

In section 1.2 we propounded a set of objectives for this work and it is the moment to rate if those have been achieved or not. The first of our objectives was to locate and study the available resources and select the ones which are easy to use and which contains reliable data. In section number 2.6 it is shown this work, therefore we can tick as acomplished and finished this objective. Our next task required to adapt each data of the resources to an internal system format so it is independent of the input. Besides, in this objective it is also proposed to facilitate as much as we can the text generation phase. Chapter 4 is dedicated exclusively to the input preprocessing before the natural language generation. We consider that with all this functionality it is described, we can convert the resources to an internal format and also to provide translations of the relations so the graph is more connected and we know more about the data. Because of the reasons we just have mentioned, we conclude our second objective is fulfilled and satisfied. As a third objective of this project, we propose the development of a description generator that is capable of showing the characteristics of the concepts of the input of the system. In chapter 5 we can find the generators we have been implementing and improving during this project, we think that several of them produce texts with enough quality to set as completed this objective. Lastly, we proposed the objective of introducing the final application of this work in the platform of the *ConCreTe* project. Because of the lack of time added to the fact that we do not still have available the tool generated in the master final project mentioned in section 2.5.2, which produces as its output the input our system is designed for, we did not added the web service into the project. We can not set as done by now this last objective.

Now we are going to mention other observations we have realised during the development of the project.

The generation of texts is not a trivial task in computing. Especially, when we want to provide a person's naturality and simulate their behaviour. To address that we have noticed we have to make use of a few computing fields, such as computional creativity or artificial intelligence, so that we can create the texts. It is also needed to check the last research it has been made to get this aim and thus we can ponder how to make an approach to our problem, besides knowing the available resources that can be a big help when obtaining data and useful functionality.

This work as a part of a larger project, *ConCreTe*, which we already talked about in section number 2.5, it has to adapt itself to the way the project works and its necesities at the same time we make a research to get to the besought goals. As a side effect of collaboration between the project *partners*, this work is partially force to use the resources and tools developed by them, it leads us to carry out a tester function for them and suggest them upgrades that improves the development which help us both.

Textifier is a tool we talked in chapter 3, it has been developed during this work as a first approach to the text generation based on input data we had in the moment in which the project was. On the one hand, from the point of view of a reader, the resulting text of th execution of the system has a good quality and it also sounds fairly well, natural, this is because the data we handled are fragments of sentences written by people, by hand. On the other hand, from the point of view of the programmer or the researcher, it is not such interesting working with that data, since it provides very little flexibility when creating the text, that reduces in a big amount the originality characteristic and creativty of the system we were lookig for. The detected limitations we found after testing the whole system revealed that it is a better starting point to use conceptual data a lot simpler, which lead us to our next objective, create from scratch a text generator adapted to the new input data. With all this in mind, a better approach from the very begining would have meant a much more quick progress of the project.

The iterative development it has been followed to create the diferent natural language generators has provided several tools to achieve different texts. Thereby is how we manage to made several proposals that were improving the less highlighted properties of the older versions. These modules have different characteristics and they have allowed us to see some of the qualities of the texts generated by people, that fact becomes useful and it is very interesting to make the output of the system seem more to what a human would generate. Moreover, we have been able to notice that the system's

architecture can be more complicated to ease the extensibility of the code, this characteristic helps and contribute to the future work, the posibilities of the generator and the quality of the output text.

Capítulo 7

Trabajo futuro

Durante el desarrollo de este trabajo se han ido planteando mejoras que no se han llevado a cabo. Otras características que influirían positivamente al sistema se proponen como reflexión final. A continuación vamos a enumerar características que se proponen como mejoras de este proyecto.

- Continuar y mejorar el estudio, explicado en el apartado 5.2, realizado durante el desarrollo del generador de textos elaborado. Sería interesante recoger muchos más textos de los dos cuestionarios que se realizaron y realizar otros cuestionarios con otros datos de entrada que utilicen un concepto con un grafo semántico con propiedades diferentes (relaciones, mayor cohesión, caminos desde el término hasta un nodo final más largos, grafos con ciclos). Con esta mejora no solo podríamos generar una plantilla mejor, pero resulta más interesante aplicar el conocimiento y los resultados extraídos a generadores como el generador con planificación simplificado.

- Para ayudar al experimento del que hemos hablado, podríamos desarrolar un software que estudiase los textos de los usuarios y sacase características comunes entre ellos. Esto podría realizarse mediante aprendizaje automático y un analizador sintáctico como *Freeling*, pero la envergadura de este objetivo puede ser propia de otro trabajo.

- Añadir otras instrucciones y utilizarlas en los generadores con planificación cuando resulten necesarias o útiles. Las instrucciones podrían generar preposiciones o conectores entre ideas, por ejemplo.

- Estructuración del texto en párrafos, agrupando ideas y secuenciando lógicamente las frases para describir los conceptos de forma más orde-

nada. Esto requeriría el uso de otros grafos semánticos más complejos de los que hemos tratado o visto en los ejemplos de este trabajo.

- Uso de grafos semánticos de otro tipo, más ricos y abundantes, para las pruebas y para comparar el comportamiento de los generadores.
- Probar a utilizar aleatoriedad en ciertas decisiones de los generadores, así conseguimos soluciones diferentes y más creativas.
- Buscar los conceptos y conseguir averiguar si existen parejas con significados opuestos, y en caso de compartir una relación coherente con el mismo concepto sujeto podríamos construir sentencias con contraste.
- Uso de sinónimos para cambiar sustantivos por otros. Esta misma funcionalidad nos puede servir para conseguir unificar relaciones en la etapa de preprocesado de la entrada.
- Por ahora solo hemos planteado la generación de textos descriptivos, pero podemos ser mucho más creativos si probamos la generación de otros tipos de textos como poesía para explicar el conocimiento contenido en los datos de entrada.
- Para los generadores de texto que tienen un *pipeline*, podemos añadir más fases de la generación de lenguaje natural, como la de la generación de expresiones de referencia u otras mencionadas en la sección 2.1.
- Utilizar la plataforma *ConCreTeFlows* para obtener conceptos nuevos generados con el blender o mezclador a partir de otros y probar los generadores. Con los resultados podríamos afinar las características de cada uno para generar textos de mayor calidad y así obtener una mejor salida en la aplicación del proyecto.
- Utilizar los grafos semánticos mejorados de los que disponemos y hemos mencionado en las contribuciones al proyecto *ConCreTe*, en el apartado 2.5.2. Así podemos ver cómo varía la calidad del texto generado dependiendo de los datos de entrada, teniendo en cuenta la fase de preprocesado de los datos que nosotros utilizamos antes de la fase de generación de texto.

Bibliografía

Amabile, T. M. The social psychology of creativity: A componential conceptualization. *Journal of personality and social psychology*, vol. 45(2), página 357, 1983.

Appelt, D. E. Planning english referring expressions. *Artificial intelligence*, vol. 26(1), páginas 1–33, 1985.

Cassell, J., Stone, M. y Yan, H. Coordination and context-dependence in the generation of embodied conversation. En *Proceedings of the first international conference on Natural language generation-Volume 14* , páginas 171–178. Association for Computational Linguistics, 2000.

Colton, S., Wiggins, G. A. et al. Computational creativity: The final frontier? En *ECAI*, vol. 12, páginas 21–26. 2012.

Concepción, E., Gervás, P. y Méndez, G. Mining knowledge in storytelling systems for narrative generation. 2016.

Cook, M. y Colton, S. Ludus ex machina: Building a 3d game designer that competes alongside humans. En *Proceedings of the 5th international conference on computational creativity*, vol. 380. 2014.

Davey, A. Discourse production. 1979.

Dybala, P., Ptaszynski, M., Maciejewski, J., Takahashi, M., Rzepka, R. y Araki, K. Multiagent system for joke generation: Humor and emotions combined in human-agent conversation. *Journal of Ambient Intelligence and Smart Environments*, vol. 2(1), páginas 31–48, 2010.

Fauconnier, G. y Turner, M. Conceptual integration networks. *Cognitive science*, vol. 22(2), páginas 133–187, 1998.

Fauconnier, G. y Turner, M. *The way we think: Conceptual blending and the mind's hidden complexities*. Basic Books, 2008.

Fellbaum, C. A semantic network of english verbs. *WordNet: An electronic lexical database*, vol. 3, páginas 153–178, 1998.

Felzenszwalb, P., McAllester, D. y Ramanan, D. A discriminatively trained, multiscale, deformable part model. En *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, páginas 1–8. IEEE, 2008.

Fleiss, J. L., Levin, B. y Paik, M. C. *Statistical methods for rates and proportions*. John Wiley & Sons, 2013.

Galván, P., Francisco, V., Hervás, R. y Méndez, G. Riddle generation using word associations. 2016a.

Galván, P., Francisco, V., Hervás, R., Méndez, G. y Gervás, P. Exploring the role of word associations in the construction of rhetorical figures. 2016b.

Gervás, P., Hervás, R., León, C. y Gale, C. V. Annotating musical theatre plots on narrative structure and emotional content. 2016.

Goldman, N. M. Conceptual generation. *Conceptual information processing*, páginas 289–371, 1975.

Jordanous, A. A standardised procedure for evaluating creative systems: Computational creativity evaluation based on what it is to be creative. *Cognitive Computation*, vol. 4(3), páginas 246–279, 2012.

Krishnamoorthy, N., Malkarnenkar, G., Mooney, R. J., Saenko, K. y Guadarrama, S. Generating natural-language video descriptions using text-mined knowledge. En *AAAI*, vol. 1, página 2. 2013.

Laptev, I., Marszalek, M., Schmid, C. y Rozenfeld, B. Learning realistic human actions from movies. En *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, páginas 1–8. IEEE, 2008.

Leskovec, J., Grobelnik, M. y Milic-Frayling, N. Learning semantic graph mapping for document summarization. En *Proceedings of ECML/PKDD-2004 Workshop on Knowledge Discovery and Ontologies*. 2004.

Leskovec, J., Milic-Frayling, N., Grobelnik, M. y Leskovec, J. Extracting summary sentences based on the document semantic graph. *Microsoft Research, Microsoft Corporation*, 2005.

McDonald, S. Exploring the process of inference generation in sarcasm: A review of normal and clinical studies. *Brain and language*, vol. 68(3), páginas 486–506, 1999.

McKeown, K. R. *Text generation: using discourse strategies and focus constraints to generate natural language text*. Cambridge University Press, 1985.

Méndez, G., Gervás, P. y León, C. On the use of character affinities for story plot generation. En *Knowledge, Information and Creativity Support Systems*, páginas 211–225. Springer, 2016a.

Méndez, G., Gervás, P. y León, C. Using fuzzy logic to model character affinities for story plot generation. 2016b.

Miller, G. A. Wordnet: a lexical database for english. *Communications of the ACM*, vol. 38(11), páginas 39–41, 1995.

Ogden, C. K. *Basic English: A general introduction with rules and grammar*. 29. K. Paul, Trench, Trubner, 1944.

Oliveira, A., Pereira, F. C. y Cardoso, A. Automatic reading and learning from text. En *Proceedings of the international symposium on artificial intelligence (ISAI)*. sn, 2001.

Reiter, E., Dale, R. y Feng, Z. *Building natural language generation systems*, vol. 33. MIT Press, 2000.

van de Riet, R. y Meersman, R. Knowledge graphs. En *Linguistic Instruments in Knowledge Engineering: Proceedings of the 1991 Workshop on Linguistic Instruments in Knowledge Engineering, Tilburg, the Netherlands, 17-18 January 1991*, página 97. North-Holland, 1992.

Rodríguez Ferreira, T. Content filtering and enrichment using triplets for text generation. 2016.

Rodriguez-Ferreira, T., Rabadán, A., Hervás, R. y Diaz, A. Improving information extraction from wikipedia texts using basic english. 2016.

Sowa, J. F. Semantic networks. encyclopedia of artificial intelligence. *Ed. Stuart C Shapiro*, 1987.

Apéndice A

Configuración de *ConCreTe Flows*

En este apéndice se muestra cómo configurar la plataforma *ConCreTe Flows* para utilizar las herramientas disponibles en esta y añadir nuevas aplicaciones. Algunos de los siguientes pasos necesitan instalar o descargar archivos, así que el tiempo necesario para completarlos puede variar y demorarse según las características del computador y conexión.

Los requisitos para el uso de *ConCreTe Flows* en un ordenador son, disponer de una versión de *Python* igual o superior a la 2.5. Debemos disponer de *pip*, por lo que debemos descargar el archivo "get-pip.py" de internet y ejecutar el comando "python get-pip.py" con permisos de administrador. También necesitaremos un cliente *Git*, lo podemos encontrar en la web [1].

Después de haber completado los pasos mencionados debemos generar una *SSH Key*, que permitirá identificar unívocamente nuestro ordenador y darla de alta en el repositorio de *ConCreTe*, esto se hace mediante la web designada a ello [2] y que está administrada por los colaboradores de Liubliana (JSI). Como paso previo a dar de alta nuestra clave, tenemos que crearnos una cuenta en el sistema, solicitándolo a dichos administradores.

Ahora, ya podemos descargar mediante *Git* el repositorio para trabajar con él de forma local, para ello podemos usar cualquiera de las interfaces disponibles (interfaz gráfica o terminal), para el caso de la terminal utilizaremos "git clone git@source.ijs.si:concrete/mothra.git".

[1] `http://git-scm.com/`

[2] `http://source.ijs.si/concrete/mothra/wikis/home`

El siguiente paso es instalar los requisitos de *mothra* (nombre del repositorio). Para ello debemos asegurarnos de que *pip* está incluido en el "path" del sistema, en caso de no estarlo debemos incluirlo. Si tenemos dificultades para encontrar el archivo, este se encuentra en la carpeta "Scripts", dentro de la carpeta donde se instaló "python". Una vez hecho esto, podemos ejecutar mediante la terminal del ordenador el comando correspondiente para situarnos en la carpeta donde descargamos "mothra" y ejecutar el comando "pip install -r requirements.txt". En este paso podemos encontrar errores, para solucionarlos podemos modificar el fichero "requirements.txt" y comentar aquellos requisitos que nos den problemas, aunque debemos tener en cuenta que si hacemos esto puede que no podamos utilizar toda la funcionalidad disponible de la plataforma. Si deseamos no modificar el archivo tendremos que descargar e instalar requisitos adicionales mediante internet según los fallos que aparezcan. Por último renombraremos el archivo situado en "mothra" llamado "__local_settings.py" por "local_settings.py" (solo es eliminar los dos caracteres "_" por los que comienza el nombre del archivo).

Para sincronizar la base de datos debemos introducir los siguientes comandos en la terminal del sistema y en el orden en el que se enumeran, en caso de que saliera un mensaje que solicita crear un superusuario diremos que no, por ahora. Los comandos a ejecutar son "python manage.py syncdb" y "python manage.py migrate". Ahora es el momento de crear un superusuario, lo haremos mediante el comando "python manage.py createsuperuser" y tendremos que proporcionar un nombre y una contraseña. Como paso final ejecutaremos "python manage.py auto_import_packages -n" que importará de forma automática los paquetes necesarios.

En este punto ya podemos ejecutar la plataforma, lo haremos con "python manage.py runserver". En caso de que queramos utilizar el modo debug ejecutaremos el servidor con el comando "python manage.py runserver_plus" en vez del anterior, en nuestro caso no va a ser necesario utilizarlo. Para acceder al servidos debemos abrir un navegador e introducir la dirección `http://127.0.0.1:8000`, donde se nos pedirá el nombre y contraseña del superusuario que nos hemos creado antes.

Debemos tener en cuenta que esto es una copia local, y que al haber muchos cambios de forma contínua en el repositorio, para acceder a nuevos recursos tendremos que volver a sincronizar la copia local haciendo "pull" con *Git*. Cada vez que hagamos esto, tendremos que volver a ejecutar el comando "python manage.py auto_import_packages" también ya que se podrían haber añadido nuevos paquetes necesarios.

Para contribuir al proyecto hay que crear un widget, existen dos formas de crearlo, desde cero (abstract widget) o mediante un servicio web. Nos

vamos a centrar en crear un widget mediante un servicio web, ya que es la forma más sencilla y más extensible de trabajar. Comenzamos creando un abstract widget y proporcionando los datos que se nos solicitan, entre los cuales aparecen el nombre, la descripción y otros datos tanto informativos como necesarios para programar la funcionalidad.

Un servicio web es accesible mediante una dirección web, por lo que tendremos que disponer de ella, y esta, según la entrada, genera una respuesta en un formato, el cual también debemos conocer (para simplificar supondremos que usamos *JSON*). Con esto datos podemos utilizar como plantilla en *python* para generar el widget el siguiente fragmento de código:

```
import urllib2
import json

input = "some input to process"
input = input.replace (" ", "+")
url = 'http://www.myweb.com/script.php?systemInput=' + input
response = urllib2.urlopen(url).read()
data = json.loads(response)
print "System's answer: " + str( data['chararcerCount'] )
```

Una vez utilizada esta plantilla para generar el código que nos interesa, ya podemos terminar la creación y utilizarlo de forma local en nuestro ordenador. A continuación ejecutamos el comando ‘‘pythonmanage.pyexport_package-uworkflows/$your_package_name$/db/package_data.json$your_package_name$’’ (donde las cadenas rodeadas con el caracter “$” indican que hay que reemplazar ese texto por el correspondiente en nuestro caso). Ahora, o cuando consideremos que el servicio está listo para que el resto de colaboradores lo puedan utilizar, tenemos que hacerlo accesible para ellos. Lo primero es asegurarnos de que tenemos la última versión del proyecto y que no vamos a sobreescribir datos más actualizados, para ello realizaremos un “pull” con *Git* mediante el comando “git pull git@source.ijs.si:concrete/mothra.git”. En caso de que se hayan descargado nuevas características volveremos a ejecutar “python manage.py auto_import_packages”. Finalmente añadimos los archivos que hemos modificado con “git add $file_path$”, y realizamos los “commit” necesarios con “git commit $file_path$” de los mismos archivos. Terminamos esta fase subiéndolo al repositorio con el comando “git push”.

Apéndice B

Improving Information Extraction from Wikipedia Texts using Basic English

Improving Information Extraction from Wikipedia Texts using Basic English

Teresa Rodríguez-Ferreira, Adrián Rabadán, Raquel Hervás, Alberto Díaz
Facultad de Informática
Universidad Complutense de Madrid
teresaro@ucm.es, arabadan@ucm.es, raquelhb@fdi.ucm.es, albertodiaz@fdi.ucm.es

Abstract

The aim of this paper is to study the effect that the use of Basic English versus common English has on information extraction from online resources. The amount of online information available to the public grows exponentially, and is potentially an excellent resource for information extraction. The problem is that this information often comes in an unstructured format, such as plain text. In order to retrieve knowledge from this type of text, it must first be analysed to find the relevant details, and the nature of the language used can greatly impact the quality of the extracted information. In this paper, we compare triplets that represent definitions or properties of concepts obtained from three online collaborative resources (English Wikipedia, Simple English Wikipedia and Simple English Wiktionary) and study the differences in the results when Basic English is used instead of common English. The results show that resources written in Basic English produce less quantity of triplets, but with higher quality.

Keywords: Information Extraction, Triplets, Basic English

1. Introduction

Although software applications could theoretically benefit from the huge amount of information in the Web, they usually face the problem of this information appearing in the form of unstructured data like plain text. The possibility of automatically extracting the knowledge underlying this plain text is therefore becoming increasingly important.

Information Extraction (IE) is the process of automatically extracting structured data from unstructured texts. There are different ways to represent data extracted from text, such as in the form of graphs or by using triplets in the form *(concept$_1$, verb, concept$_2$)* to express relations between concepts extracted from the text. Although there are many IE approaches, in this paper we are only interested in unsupervised techniques that are able to extract information from plain text. For this kind of technique, the characteristics of the source text from which the information is going to be extracted play an important role in the obtained results.

In this paper we will evaluate whether the use of Basic English instead of common English leads to the extraction of more accurate data by implementing an experiment that compares triplets extracted from the English Wikipedia[1], Simple English Wikipedia[2] and Simple English Wiktionary[3] (from now on referred to as Simple Wikipedia and Simple Wiktionary). Basic English is a simplification of the English Language created by Ogden (1930) which defends that full communication can be achieved by using only 850 English words. In addition to using Basic English, Simple Wikipedia and Simple Wiktionary also ask users to write in shorter sentences, use active voice over passive voice and provide guidelines to help users write sentences with simple structures.

The triplets used will represent definitions and properties, concepts that establish a unidirectional IS_A or IS relation with certain other concepts. Even though these two relations are different, they can both be used to define a concept, so they have not been considered separately in the final results. This type of output will be easily computable by machines and can be used to establish new relations between concepts. This can be achieved, for instance, by connecting triplets in which the second concept is the same as the first concept of the other triplet.

The paper will address questions such as:

- Are triplets obtained from text written in Basic English more useful?
- How does information obtained from dictionaries compare to information obtained from encyclopedias?

The goal of this work is not to provide a new IE technique that improves previous work results, but to demonstrate that texts written using simplified vocabulary and grammar will lead to better triplet extraction.

In Section 2 we discuss previous work that is relevant to the field of Information Extraction. In Section 3 we describe the sources used and the results we expect to obtain from them, and we cover implementation details. In Section 4 we explain the evaluation criteria for the quality of the triplets obtained, we present the final results and we cover the issues encountered during this research. Section 5 is a discussion of the results. Finally, Section 6 describes future work that will improve the triplet extraction system.

2. Related work

Information Extraction (IE), the process of automatically extracting structured information from unstructured texts, has progressed substantially over the last few decades (Etzioni et al., 2008). Although the ambiguous nature of plain text makes the task an arduous one, it is possible to find many systems that have obtained quite good results. TextRunner (Yates et al., 2007), one of the pioneers in Open Information Extraction (OIE), is able to obtain high-quality information from text in a scalable and general manner.

[1] http://www.wikipedia.org
[2] http://simple.wikipedia.org
[3] http://simple.wiktionary.org

Rusu et al. (2007) present an approach to extracting triplets from sentences by relying on well known syntactic parsers for English.
Wikipedia is considered an excellent source of texts for IE systems due to its broad variety of topics and advantageous characteristics such as the quality of the texts and their internal structure. Therefore there are some IE systems that work with Wikipedia texts and/or their structured metadata, like Wanderlust (Akbik and Bross, 2009) or WOE (Wikipedia-based Open Extractor) (Wu and Weld, 2010). Weld et al. (2009) restrict their process to infoboxes, tabular summaries of an article's salient details which are included in a number of Wikipedia pages. Wanderlust (Akbik and Bross, 2009) is an algorithm that automatically extracts semantic relations from natural language text. The procedure uses deep linguistic patterns that are defined over the dependency grammar of sentences. Due to its linguistic nature, the method performs in an unsupervised fashion and is not restricted to any specific type of semantic relation. The applicability of the algorithm is tested using the English Wikipedia corpus. WOE (Wikipedia-based Open Extractor) (Wu and Weld, 2010) is a system capable of using knowledge extracted from a heuristic match between Wikipedia infoboxes and corresponding text. In particular, Krawczyk et al. (2015) present a method of acquiring new ConceptNet triplets automatically extracted from Japanese Wikipedia XML dump files. In order to check the validity of their method, they used human annotators to evaluate the quality of the obtained triplets.

3. Using Basic English for improving Information Extraction from texts

Our goal is to extract triplets which represent definitions or properties of a given concept established by a unidirectional IS_A or IS relation. Many other relations can be considered, but they are out of the scope of this experiment.

3.1. Textual knowledge sources

The sources where the triplets are extracted from must contain definitions and properties of concepts. The most appropriate resources for this purpose are dictionaries and encyclopedias. Dictionaries provide succinct definitions and a brief and usually more technical overview of the concept's most salient properties. Encyclopedias, on the other hand, contain more general information and in greater quantity.
We have chosen to use Wikipedia, Simple Wikipedia and Simple Wiktionary as sources for Information Extraction. All three are free-access and free-content collaborative Internet encyclopedias or dictionaries. This type of resource is fast-growing, with content created by users from all over the world (refer to Table 1).
Wikipedia is ranked as one of the top ten most popular websites at the time this article is written, so it provides a rich source of general reference information for this type of work. One of the main concerns when using a free-content resource is the quality of its content and language. Since we are not going to attempt to extract complex details of the concepts, the accuracy of these sources does not pose an impediment, because their general definitions tend to be correct. On the other hand, the structure of the text can be problematic when parsing the information. A simple grammatical error or an incorrectly structured sentence may lead to no triplets being extracted, or to triplets containing properties which are not definitions of the concept. This type of error is more likely to occur in sources where articles are longer and more complex.
Below is an example of a fragment of text extracted from the same article for each of the different sources:

1. Wikipedia: "Chocolate is a typically sweet, usually brown, food preparation of Theobroma cacao seeds, roasted and ground, often flavored, as with vanilla. It is made in the form of a liquid, paste, or in a block, or used as a flavoring ingredient in other foods."

2. Simple Wikipedia: "Chocolate is a food made from the seeds of a cacao tree. It is used in many desserts like pudding, cakes, candy, and ice cream. It can be a solid form like a candy bar or it can be in a liquid form like hot chocolate."

3. Simple Wiktionary: "Chocolate is a candy made from cacao beans and often used to flavour other foods such as cakes and cookies. A chocolate is an individual candy that is made of or covered in chocolate. Chocolate is a dark brown colour."

3.2. Triplet extraction

In order to extract relevant semantic information from the text, it must first go through a process of morphological analysis and dependency parsing. The analyser used was Freeling 2.2 (Carreras et al., 2004), an open source language analysis tool suite that supports several languages, including English.
The information for each specified concept was obtained from the corresponding web page from each source. For example, for the concept *pinneaple* and the source Simple Wikipedia the wiki page used was *https://simple.wikipedia.org/wiki/Pineapple*. This information was parsed into plain text, and then morphologically analysed using Freeling 2.2 (Carreras et al., 2004). This was in turn used as input for the dependency parsing, producing a final output of a tree containing all the semantic information. After this, the objective was to extract only IS_A or IS relations from the texts, so only sentences which had as their root any form of the verb "to be" were considered. Assertions that make use of a form other than the present tense were taken into consideration because texts referring to historic events or characters may use the past tense. Once the relevant sentences had been collected, the next step was to find the ones referring to the specified concept. Since the aim is to extract IS_A or IS relations, the third element of the triplets is always a definition or a property of the first element, so the triplets follow this structure: *concept - verb - property*.
In order to obtain definitions of the concept or related information from the text, the object of the chosen sentences has been studied. There are three possible scenarios depending on the root of the object (refer to Table 2):

1. When the root of the object is a noun, it is considered as a possible definition of the concept. For instance

-	English Wikipedia	Simple Wikipedia	Simple Wiktionary
Articles	4,977,081	115,138	24,309
Users	26,395,232	470,736	14,981
Articles per user	0.19	0.24	1.62

Table 1: Usage statistics of the used resources

in the sentence "A pineapple is a fruit", the object is "a fruit" and its root is "fruit", which is a noun, so it is saved in a triplet (pineapple - be - fruit). This represents an IS_A relation.

2. If the noun has any modifiers which are adjectives, they are also selected as possible information related to the concept. For instance in the phrase: "Chocolate is a dark brown colour", the root of the object ("colour") has two modifiers, "dark" and "brown", so aside from the triplet that represents an IS_A relation (chocolate - be - colour), both adjectives are stored in additional triplets (chocolate - be - dark, chocolate - be - brown). This type of information represents a property of the concept, an IS relation.

3. If the root of the object is the conjunction "and" or "or" instead of a noun, its children are searched for nouns and adjectives much like in the previous case, for example in the sentence "Battle Royale is a novel and a film" (Battle_Royale - be - novel, Battle_Royale - be - film). This represents an IS_A relation when the child is a noun or an IS relation when it is an adjective.

As an example, we can observe the differences between the properties extracted for the concept "wine":

- From Wikipedia, the extracted properties for the triplets were *cabernet_sauvignon*, *gamay*, *merlot*, *part*, *tradition* and *red*.
- From Simple Wikipedia, the properties were *drink*, *alcoholic* and *popular*.
- From Simple Wiktionary, only one property was extracted: *drink*.

4. Evaluation

The evaluation criteria used to verify the quality of the extracted triplets is similar to the one used by Krawczyk et al. (2015). Every triplet generated for each concept is assigned a value based on how strongly related its property is to the concept and how well it respects the relation. The possible values are 1, 0.5 and 0.

- Triplets get the highest score when they correctly represent an IS_A or IS relation in which the property defines or is very strongly related to the concept. For instance the triplet *car - be - vehicle* would be considered a good triplet and it would be assigned 1 point.
- Mediocre triplets are assigned 0.5 points, when the property is a less accurate or informative definition of the concept, or when it represents a feature or quality of the concept. Note that the IS_A or IS relation must still be respected. A triplet such as *book - be - product* would have a score of 0.5 points.
- Triplets with properties which are related to the concept but do not respect the relation (for example *moon - be - crater*) or which are unrelated to the concept (*chocolate - be - iron*) are considered bad triplets and receive the lowest score (0).

The evaluation so far has been performed manually by four human annotators. The triplets generated for this evaluation were divided into four groups, where each annotator evaluated two groups and each triplet was evaluated by two annotators. The final statistics were obtained by using the average of the score given by all of the annotators, following an inter-annotator agreement using a popular metric, Fleiss Kappa (Fleiss, 1981). This allows us to know the degree of agreement between the annotators.

4.1. Results

A total of 62 concepts were randomly chosen as input (e.g.: pinneaple, chocolate, Battle Royale...), 49 of which generated triplets for at least one of the knowledge sources. The absence of triplets for some concepts is due to texts with sentences defining the concept which do not match the required pattern accepted by the extractor. Both common nouns (water, yellow, chair...) and proper nouns (New York, Bruce Willis, Final Fantasy...) were used as input, and the latter produced less triplets (7 of the 13 concepts that did not generate any triplets were proper nouns). A total of 604 triplets were examined (428 from Wikipedia, 124 from Simple Wikipedia and 52 from Simple Wiktionary).
The results reflected in Table 3 show that sources with a large amount of content produce triplets for more concepts, as was expected. Consequently, Wikipedia is the source that offers the most good triplets (those assigned 1 point), followed by Simple Wikipedia and Simple Wiktionary. Note however that it also produces more mediocre triplets (0.5 points) and many more bad triplets (0 points) than the others. Even though the quantity of the triplets generated for sources using Basic English is compromised, their quality is much higher. Less than a third of the triplets extracted from Wikipedia can be considered good, and less than 10% are mediocre. This means that around 64% are bad triplets, representing information that is not related to the specified concepts or that does not represent an IS_A or IS relation. Triplets extracted from Simple Wikipedia behave better, more than 40% of them are good, and less than half are bad. As shown in Table 3, the degree of agreement between triplets extracted from Wikipedia and Simple Wikipedia is more or less the same. The Kappa score for Simple Wiktionary is better and shows that the annotators

Sentence	Freeling V2.2 tree	Triplets
A pineapple is a fruit	claus/top/(is be VBZ -) [n-chunk/ncsubj/(Pineapple pineapple NN -) sn-chunk/dobj/(fruit fruit NN -) [DT/det/(a a DT -)]]	Pineapple - be - fruit
Chocolate is a dark brown colour	claus/top/(is be VBZ -) [n-chunk/ncsubj/(Chocolate chocolate NN -) sn-chunk/dobj/(colour colour NN -) [DT/det/(a a DT -) attrib/ncmod/(dark dark JJ -) attrib/ncmod/(brown brown JJ -)]]	Chocolate - be - dark Chocolate - be - brown Chocolate - be - colour
Battle Royale is a novel and a film	claus/top/(is be VBZ -) [n-chunk/ncsubj/(Royale royale NNP -) [NN/ncmod/(Battle battle NN -)] sn-coor/dobj/(and and CC -) [sn-chunk/conj/(novel novel NN -) [DT/det/(a a DT -)] sn-chunk/conj/(film film NN -) [DT/det/(a a DT -)]]]	Battle_Royale - be - novel Battle_Royale - be - film

Table 2: Triplet extraction scenarios

agree more on the quality of these triplets. Since the average score is higher for this source, this proves that triplets extracted from Simple Wiktionary have an overall better quality than the others.

The amount of concepts that generated triplets was similar for both Wikipedia and Simple Wikipedia, which means that the main difference between them was the content of the text. This proves that text expressed in Basic English yields more useful definitions for concepts than text written in common English.

Finally, the best results are achieved in Simple Wiktionary. Around 55% of the generated triplets are good definitions of the concepts, slightly less than 20% are mediocre, and less than a third of the triplets are bad. This seems to indicate that sources which contain less detailed and more specific content tend to result in higher quality triplets. Dictionaries are ideal, since they strive to define concepts briefly and do not offer additional background information.

	Wikipedia	Simple Wikipedia	Simple Wiktionary
Concepts with triplets	46 (74.19%)	40 (64.52%)	26 (41.94%)
Triplets	428	124	52
Good triplets	119 (27.8%)	54.5 (43.95%)	28.5 (54.81%)
Mediocre triplets	36.5 (8.53%)	12.5 (10.08%)	9 (17.31%)
Bad triplets	272.5 (63.67%)	57 (45.97%)	14.5 (27.88%)
Average score	0.32	0.49	0.63
Inter-annotator agreement (kappa)	0.496	0.49	0.578

Table 3: Results from the evaluation

4.2. Detected errors in triplet extraction

The above method is relatively simple to understand and to implement, but it has a few disadvantages. When the text does not have any sentences that match the required pattern exactly, no triplets can be extracted. For instance, if a definition uses a verb other than "to be", but equivalent to it, the sentence will be ignored. The definition of "purple" extracted from the Wikipedia ("Purple is defined as a deep, rich shade between crimson and violet [...]") cannot be processed because "defined" is the main verb and "is" is an auxiliary verb. If the word "is" had been used by itself, the triplets *purple - be - shade, purple - be - deep* and *purple - be - rich* could have been extracted.

As explained above, when the object's root is a noun with an adjective that refers to it, both noun and adjective are

stored separately in different triplets. In some cases the concept's definition only makes sense when the adjective and noun are used together. For example, when defining a foot, the sentence "anatomical structure" was obtained. This makes sense as a combination, but a person would not usually describe a foot as a structure. When this situation arises, both words usually make sense separately as well as combined, but in some cases storing them separately renders one or both of them useless. The final decision was to keep the information separately in the triplets, ensuring that the results will be more easily computable, at the expense of having triplets which are more general and less precise.

Accepting any form of the verb "to be", including past tense, means that relevant information can be extracted from text regarding past events or historical characters. The problem is that this could also result in out of date information. For instance the sentence "In ancient times Germany was largely pagan" results in the triplet *Germany - be - pagan*. This is not true at present, and so this triplet is incorrect.

An interesting phenomenon that occurs is when providing examples of a concept. Sentences such as "Examples of [concept] are..." or "A type of [concept] could be..." match the pattern recognised by the triplet extractor, so the sentence "A popular toy of this type is the Teddy Bear" will result in the triplet *toy - be - teddy_bear*. This represents information that is related to the concept, but since it does not match the IS_A or IS relation, it cannot be considered correct.

Another problem presents itself in articles about people or characters. Sometimes they are referred to in different ways inside the article, for instance by their full name, just their first name, just their surname or even a nickname. When searching for "Bruce Willis" in the Wikipedia, he is referred to as "Walter Bruce Willis" and further ahead as just "Willis". In this case only the sentences that contain the concept written exactly as specified can be examined.

5. Conclusions

The results discussed in section 4.1. reveal that sources written in Basic English produce less quantity of triplets for a given concept than those written in English, but the triplets display much higher quality. Overall, the triplets extracted from Simple Wiktionary are twice as good as those extracted from Wikipedia. Generally, longer articles which tend to be more detailed and provide background information about the concept result in more incorrect triplets. This can be observed especially in articles concerning very general topics or articles on historical events and characters, for instance in the article regarding the Earth. For this reason, articles from Wikipedia, which are usually longer than those in Simple Wikipedia, produce more triplets per concept, but a large portion are incorrect. On the other hand, certain types of articles do not produce any triplets, especially articles regarding proper nouns (such as countries, cities, books, films, games or names of people). In our evaluation 15 concepts which are proper nouns were introduced, and roughly half of them (7) did not generate triplets for any of the sources.

The precise and succinct style of dictionaries seems more useful in the extraction of IS_A and IS relations between concepts and their properties. The triplets extracted from this type of source are also more easily evaluated by human annotators, since the information they contain is more objective. More research is needed, however, in order to correctly compare results extracted from encyclopedias against results extracted from dictionaries.

6. Future work

In this research, our goal was to compare triplets obtained from sources written in common English with those from sources written in Basic English. For this reason Wikipedia and Simple Wikipedia were the first two options to be considered. While analysing the results obtained, it seemed likely that Simple Wiktionary might be an even better source than Simple Wikipedia. This was on the grounds that aside from using Basic English and simpler sentence structure, its content is more precise and focuses solely on definitions, which was the goal of this study. We did not, however, evaluate results obtained from the English Wiktionary. It would be interesting to compare Simple Wiktionary against Wiktionary to examine the effect of IE from dictionaries written in Basic English, and to compare Wikipedia against Wiktionary to further observe the differences between data extracted from dictionaries and from encyclopedias. However, these resources could also be combined since the information contained in each one complements the others.

The extracted triplets follow a simple structure: *concept - verb - property*. In this work the verb used is always "to be", but this could be extended to also include relations such as HAS_A or RELATED_TO.

Having encountered the errors discussed in section 4.2., it would be useful to detect the patterns that lead to these errors and address them before saving the triplets. The matter of storing nouns and the adjectives that apply to them separately or together should also be explored further. When stored separately they lead to a larger amount of simpler triplets, but some information can be lost in the process, leaving either the noun or the adjective meaningless without its partner. Finally, the use of synonyms can aid in the recognition of additional triplets in the content. When searching for a concept, definitions that refer to it with a synonym (or a nickname or alternative name in the case of a person) are currently ignored. Using synonyms for common names, or alternative names found for people in sources such as DBpedia could produce richer results.

In order to reduce the time employed in the evaluation of the generated triplets, an automatic or semi-automatic criteria for evaluation should be implemented. By using existing triplets or relations similar to ours from sources such as ConceptNet, we could compare the results with others that are accepted as correct to automatically approve the common triplets.

7. Acknowledgements

This work is funded by ConCreTe. The project ConCreTe acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European

Commission, under FET grant number 611733.

8. References

Akbik, A. and Bross, J. (2009). Wanderlust: Extracting semantic relations from natural language text using dependency grammar patterns. In *Proceedings of the 2009 Semantic Search Workshop at the 18th International World Wide Web Conference*, pages 6–15, Madrid, Spain.

Carreras, X., Chao, I., Padró, L., and Padró, M. (2004). Freeling: An open-source suite of language analyzers. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC'04)*.

Etzioni, O., Banko, M., Soderland, S., and Weld, D. S. (2008). Open information extraction from the web. *Commun. ACM*, 51(12):68–74, December.

Fleiss, J. (1981). *Statistical methods for rates and proportions*. Wiley, New York, USA.

Krawczyk, M., Rzepka, R., and Araki, K. (2015). Extracting conceptnet knowledge triplets from japanese wikipedia. In *21st Annual Meeting of The Association for Natural Language Processing (NLP-2015)*, pages 1052–1055, Kyoto, Japan.

Ogden, C. K. (1930). *Basic English: A General Introduction with Rules and Grammar*. Paul Treber & Co., Ltd, London.

Rusu, D., Dali, L., Fortuna, B., Grobelnik, M., and Mladenic, D. (2007). Triplet extraction from sentences. In *Proceedings of the 10th International Multiconference Information Society*, pages 8–12.

Weld, D. S., Hoffmann, R., and Wu, F. (2009). Using wikipedia to bootstrap open information extraction. *SIGMOD Rec.*, 37(4):62–68, March.

Wu, F. and Weld, D. S. (2010). Open information extraction using wikipedia. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 118–127, Stroudsburg, PA, USA. Association for Computational Linguistics.

Yates, A., Cafarella, M., Banko, M., Etzioni, O., Broadhead, M., and Soderland, S. (2007). Textrunner: Open information extraction on the web. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, NAACL-Demonstrations '07, pages 25–26, Stroudsburg, PA, USA. Association for Computational Linguistics.

Apéndice C

Resultado de consulta a ConceptNet 5

Aquí tenemos el resultado de la consulta al recurso ConceptNet 5, donde el concepto por el que se pregunta es “actor”.

```
{
  "edges": [
    {
      "context": "/ctx/all",
      "dataset": "/d/wordnet/3.0",
      "end": "/c/en/actor/n/a_person_who_acts_and_gets_things_done",
      "features": [
        "/c/en/powerhouse/n/a_highly_energetic_and_indefatigable_person
             /r/IsA -",
        "/c/en/powerhouse/n/a_highly_energetic_and_indefatigable_person
             - /c/en/actor/n/a_person_who_acts_and_gets_things_done",
        "- /r/IsA
             /c/en/actor/n/a_person_who_acts_and_gets_things_done"
      ],
      "id": "/e/8f34715e2a7e2ce3a85517b4daad63249eb73815",
      "license": "/l/CC/By-SA",
      "rel": "/r/IsA",
      "source_uri": "/s/wordnet/3.0",
      "sources": [
        "/s/wordnet/3.0"
      ],
      "start":
          "/c/en/powerhouse/n/a_highly_energetic_and_indefatigable_person",
      "surfaceText": null,
```

```
    "uri":
        "/a/[/r/IsA/,/c/en/powerhouse/n/a_highly_energetic_and_indefatigable_person/
         ,/c/en/actor/n/a_person_who_acts_and_gets_things_done/]",
    "weight": 1.5849625007211563
  },
  {
    "context": "/ctx/all",
    "dataset": "/d/wordnet/3.0",
    "end": "/c/en/actor/n/a_person_who_acts_and_gets_things_done",
    "features": [
      "/c/en/act/v/discharge_one's_duties /r/DerivedFrom -",
      "/c/en/act/v/discharge_one's_duties -
          /c/en/actor/n/a_person_who_acts_and_gets_things_done",
      "- /r/DerivedFrom
          /c/en/actor/n/a_person_who_acts_and_gets_things_done"
    ],
    "id": "/e/3116446e16d89b02bf3e4a9237761f706545d221",
    "license": "/l/CC/By-SA",
    "rel": "/r/DerivedFrom",
    "source_uri": "/s/wordnet/3.0",
    "sources": [
      "/s/wordnet/3.0"
    ],
    "start": "/c/en/act/v/discharge_one's_duties",
    "surfaceText": null,
    "uri": "/a/[/r/DerivedFrom/,/c/en/act/v/discharge_one's_duties/
        ,/c/en/actor/n/a_person_who_acts_and_gets_things_done/]",
    "weight": 1.5849625007211563
  },
  {
    "context": "/ctx/all",
    "dataset": "/d/wordnet/3.0",
    "end": "/c/en/actor/n/a_person_who_acts_and_gets_things_done",
    "features": [
      "/c/en/do/v/get_done /r/DerivedFrom -",
      "/c/en/do/v/get_done -
          /c/en/actor/n/a_person_who_acts_and_gets_things_done",
      "- /r/DerivedFrom
          /c/en/actor/n/a_person_who_acts_and_gets_things_done"
    ],
    "id": "/e/ec35eb5474df9b9370509b4b1e94a76325b39cd4",
    "license": "/l/CC/By-SA",
    "rel": "/r/DerivedFrom",
    "source_uri": "/s/wordnet/3.0",
    "sources": [
      "/s/wordnet/3.0"
    ],
    "start": "/c/en/do/v/get_done",
    "surfaceText": null,
```

```
    "uri": "/a/[/r/DerivedFrom/,/c/en/do/v/get_done/
           ,/c/en/actor/n/a_person_who_acts_and_gets_things_done/]",
    "weight": 1.5849625007211563
  },
  {
    "context": "/ctx/all",
    "dataset": "/d/wordnet/3.0",
    "end": "/c/en/actor/n/a_person_who_acts_and_gets_things_done",
    "features": [
      "/c/en/man_of_action/n/someone_inclined_to_act_first_and_think_later
           /r/IsA -",
      "/c/en/man_of_action/n/someone_inclined_to_act_first_and_think_later
           - /c/en/actor/n/a_person_who_acts_and_gets_things_done",
      "- /r/IsA
           /c/en/actor/n/a_person_who_acts_and_gets_things_done"
    ],
    "id": "/e/9ef26898065b262e9fb52eb13f71907cf613ee9e",
    "license": "/l/CC/By-SA",
    "rel": "/r/IsA",
    "source_uri": "/s/wordnet/3.0",
    "sources": [
      "/s/wordnet/3.0"
    ],
    "start":
        "/c/en/man_of_action/n/someone_inclined_to_act_first_and_think_later",
    "surfaceText": null,
    "uri": "/a/[/r/IsA/,/c/en/man_of_action/n/
          someone_inclined_to_act_first_and_think_later/,/c/en/actor/n/
          a_person_who_acts_and_gets_things_done/]",
    "weight": 1.5849625007211563
  },
  {
    "context": "/ctx/all",
    "dataset": "/d/wordnet/3.0",
    "end": "/c/en/actor/n/a_person_who_acts_and_gets_things_done",
    "features": [
      "/c/en/energizer/n/
       someone_who_imparts_energy_and_vitality_and_spirit_to_other_people
           /r/IsA -",
      "/c/en/energizer/n/
       someone_who_imparts_energy_and_vitality_and_spirit_to_other_people
           - /c/en/actor/n/a_person_who_acts_and_gets_things_done",
      "- /r/IsA
           /c/en/actor/n/a_person_who_acts_and_gets_things_done"
    ],
    "id": "/e/38802e1c5f17975e65d6cdb4884db64b798134cc",
    "license": "/l/CC/By-SA",
    "rel": "/r/IsA",
    "source_uri": "/s/wordnet/3.0",
```

```
    "sources": [
      "/s/wordnet/3.0"
    ],
    "start": "/c/en/energizer/n/
          someone_who_imparts_energy_and_vitality_and_spirit_to_other_people",
    "surfaceText": null,
    "uri": "/a/[/r/IsA/,/c/en/energizer/n/
          someone_who_imparts_energy_and_vitality_and_spirit_to_other_people/
          ,/c/en/actor/n/a_person_who_acts_and_gets_things_done/]",
    "weight": 1.5849625007211563
  },
  {
    "context": "/ctx/all",
    "dataset": "/d/wordnet/3.0",
    "end": "/c/en/actor/n/a_person_who_acts_and_gets_things_done",
    "features": [
      "/c/en/go-getter/n/someone_whose_career_progresses_rapidly
           /r/IsA -",
      "/c/en/go-getter/n/someone_whose_career_progresses_rapidly -
           /c/en/actor/n/a_person_who_acts_and_gets_things_done",
      "- /r/IsA
           /c/en/actor/n/a_person_who_acts_and_gets_things_done"
    ],
    "id": "/e/b77de9f340e07687bac7fa77b8ff91f92d2236c5",
    "license": "/l/CC/By-SA",
    "rel": "/r/IsA",
    "source_uri": "/s/wordnet/3.0",
    "sources": [
      "/s/wordnet/3.0"
    ],
    "start":
        "/c/en/go-getter/n/someone_whose_career_progresses_rapidly",
    "surfaceText": null,
    "uri":
        "/a/[/r/IsA/,/c/en/go-getter/n/someone_whose_career_progresses_rapidly/
          ,/c/en/actor/n/a_person_who_acts_and_gets_things_done/]",
    "weight": 1.5849625007211563
  },
  {
    "context": "/ctx/all",
    "dataset": "/d/wordnet/3.0",
    "end": "/c/en/person/n/a_human_being",
    "features": [
      "/c/en/actor/n/a_person_who_acts_and_gets_things_done /r/IsA
           -",
      "/c/en/actor/n/a_person_who_acts_and_gets_things_done -
           /c/en/person/n/a_human_being",
      "- /r/IsA /c/en/person/n/a_human_being"
    ],
```

```
    "id": "/e/a9cc0f76a21a1bdfce07d15127ea3536f81c6136",
    "license": "/l/CC/By-SA",
    "rel": "/r/IsA",
    "source_uri": "/s/wordnet/3.0",
    "sources": [
      "/s/wordnet/3.0"
    ],
    "start": "/c/en/actor/n/a_person_who_acts_and_gets_things_done",
    "surfaceText": null,
    "uri":
        "/a/[/r/IsA/,/c/en/actor/n/a_person_who_acts_and_gets_things_done/
         ,/c/en/person/n/a_human_being/]",
    "weight": 1.5849625007211563
  },
  {
    "context": "/ctx/all",
    "dataset": "/d/wordnet/3.0",
    "end": "/c/en/actor/n/a_person_who_acts_and_gets_things_done",
    "features": [
      "/c/en/perform/v/carry_out_or_perform_an_action /r/DerivedFrom
          -",
      "/c/en/perform/v/carry_out_or_perform_an_action - /c/en/actor/n/
         a_person_who_acts_and_gets_things_done",
      "- /r/DerivedFrom
          /c/en/actor/n/a_person_who_acts_and_gets_things_done"
    ],
    "id": "/e/1f74f4e13428a994ddd51657f905473cfdbd93fc",
    "license": "/l/CC/By-SA",
    "rel": "/r/DerivedFrom",
    "source_uri": "/s/wordnet/3.0",
    "sources": [
      "/s/wordnet/3.0"
    ],
    "start": "/c/en/perform/v/carry_out_or_perform_an_action",
    "surfaceText": null,
    "uri":
        "/a/[/r/DerivedFrom/,/c/en/perform/v/carry_out_or_perform_an_action/
         ,/c/en/actor/n/a_person_who_acts_and_gets_things_done/]",
    "weight": 1.5849625007211563
  },
  {
    "context": "/ctx/all",
    "dataset": "/d/wordnet/3.0",
    "end": "/c/en/actor/n/a_person_who_acts_and_gets_things_done",
    "features": [
      "/c/en/eager_beaver/n/an_alert_and_energetic_person /r/IsA -",
      "/c/en/eager_beaver/n/an_alert_and_energetic_person -
          /c/en/actor/n/a_person_who_acts_and_gets_things_done",
```

```
      "- /r/IsA
          /c/en/actor/n/a_person_who_acts_and_gets_things_done"
    ],
    "id": "/e/9eb3c9302dcc9f2fa7e6066fe9f2e41ba532ef37",
    "license": "/l/CC/By-SA",
    "rel": "/r/IsA",
    "source_uri": "/s/wordnet/3.0",
    "sources": [
      "/s/wordnet/3.0"
    ],
    "start": "/c/en/eager_beaver/n/an_alert_and_energetic_person",
    "surfaceText": null,
    "uri":
        "/a/[/r/IsA/,/c/en/eager_beaver/n/an_alert_and_energetic_person/
         ,/c/en/actor/n/a_person_who_acts_and_gets_things_done/]",
    "weight": 1.5849625007211563
  },
  {
    "context": "/ctx/all",
    "dataset": "/d/wordnet/3.0",
    "end": "/c/en/actor/n/a_person_who_acts_and_gets_things_done",
    "features": [
      "/c/en/work/v/exert_oneself_by_doing_mental_or_physical_work_for
       _a_purpose_or_out_of_necessity /r/DerivedFrom -",
      "/c/en/work/v/exert_oneself_by_doing_mental_or_physical_work_for
       _a_purpose_or_out_of_necessity - /c/en/actor/n/a_person_who_
          acts_and_gets_things_done",
      "- /r/DerivedFrom
          /c/en/actor/n/a_person_who_acts_and_gets_things_done"
    ],
    "id": "/e/8b20bd770a18ca17b752aef2c7b27c7fa1568f0a",
    "license": "/l/CC/By-SA",
    "rel": "/r/DerivedFrom",
    "source_uri": "/s/wordnet/3.0",
    "sources": [
      "/s/wordnet/3.0"
    ],
    "start": "/c/en/work/v/exert_oneself_by_doing_mental_or_physical_
        work_for_a_purpose_or_out_of_necessity",
    "surfaceText": null,
    "uri": "/a/[/r/DerivedFrom/,/c/en/work/v/
    exert_oneself_by_doing_mental_or_physical_work_for_a_purpose_
       or_out_of_necessity/,/c/en/actor/n/
          a_person_who_acts_and_gets_things_done/]",
    "weight": 1.5849625007211563
  },
  {
    "context": "/ctx/all",
    "dataset": "/d/wordnet/3.0",
```

```
      "end": "/c/en/actor/n/a_person_who_acts_and_gets_things_done",
      "features": [
        "/c/en/demon/n/someone_extremely_diligent_or_skillful /r/IsA -",
        "/c/en/demon/n/someone_extremely_diligent_or_skillful -
             /c/en/actor/n/a_person_who_acts_and_gets_things_done",
        "- /r/IsA
             /c/en/actor/n/a_person_who_acts_and_gets_things_done"
      ],
      "id": "/e/c0af0826a966ed2294f87a32f1bb874e1321fd00",
      "license": "/l/CC/By-SA",
      "rel": "/r/IsA",
      "source_uri": "/s/wordnet/3.0",
      "sources": [
        "/s/wordnet/3.0"
      ],
      "start": "/c/en/demon/n/someone_extremely_diligent_or_skillful",
      "surfaceText": null,
      "uri":
          "/a/[/r/IsA/,/c/en/demon/n/someone_extremely_diligent_or_skillful/
           ,/c/en/actor/n/a_person_who_acts_and_gets_things_done/]",
      "weight": 1.5849625007211563
    }
  ],
  "numFound": 11
}
```

Apéndice D

Resultado de consulta a Thesaurus Rex

En este apéndice podemos ver el resultado de la consulta al recurso Thesaurus Rex, donde el concepto por el que se pregunta es “cafe”.

```
<MemberData>
  <Categories member="cafe">
    <Category weight="10">shared:facility</Category>
    <Category weight="183">convenient:location</Category>
    <Category weight="145">public:location</Category>
    <Category weight="40">communal:facility</Category>
    <Category weight="5">suitable:location</Category>
    <Category weight="6">local:establishment</Category>
    <Category weight="13">cool:stuff</Category>
    <Category weight="20">key:location</Category>
    <Category weight="67">commercial:location</Category>
    <Category weight="48">communal:area</Category>
    <Category weight="5">large:object</Category>
    <Category weight="58">public:facility</Category>
    <Category weight="11">indoor:facility</Category>
    <Category weight="18">physical:area</Category>
    <Category weight="19">stationary:area</Category>
    <Category weight="45">social:facility</Category>
    <Category weight="14">recreational:facility</Category>
    <Category weight="18">outdoor:area</Category>
    <Category weight="13">sufficient:facility</Category>
    <Category weight="44">public:area</Category>
    <Category weight="7">friendly:facility</Category>
    <Category weight="13">fixed:installation</Category>
    <Category weight="9">communal:location</Category>
```

```
    <Category weight="23">local:facility</Category>
    <Category weight="12">private:location</Category>
    <Category weight="15">busy:area</Category>
    <Category weight="11">modern:facility</Category>
    <Category weight="5">related:facility</Category>
    <Category weight="13">tangible:facility</Category>
    <Category weight="46">neutral:location</Category>
    <Category weight="13">major:location</Category>
    <Category weight="13">secure:location</Category>
    <Category weight="41">good:facility</Category>
    <Category weight="48">remote:location</Category>
    <Category weight="33">indoor:location</Category>
    <Category weight="10">key:facility</Category>
    <Category weight="5">real:location</Category>
    <Category weight="6">informal:establishment</Category>
    <Category weight="40">outdoor:location</Category>
    <Category weight="14">civilian:facility</Category>
    <Category weight="19">accessible:facility</Category>
    <Category weight="4">public:establishment</Category>
    <Category weight="33">commercial:area</Category>
    <Category weight="9">common:facility</Category>
    <Category weight="16">accessible:location</Category>
    <Category weight="4">public:structure</Category>
    <Category weight="33">elaborate:facility</Category>
    <Category weight="14">private:facility</Category>
    <Category weight="6">informal:area</Category>
    <Category weight="10">extra:facility</Category>
    <Category weight="36">public:arena</Category>
    <Category weight="23">new:facility</Category>
    <Category weight="31">convenient:facility</Category>
    <Category weight="12">nearby:facility</Category>
    <Category weight="8">ancillary:facility</Category>
    <Category weight="15">industrial:unit</Category>
    <Category weight="15">urban:facility</Category>
    <Category weight="7">populated:area</Category>
    <Category weight="14">poor:facility</Category>
    <Category weight="9">large:location</Category>
    <Category weight="31">physical:location</Category>
    <Category weight="8">commercial:facility</Category>
    <Category weight="40">private:building</Category>
    <Category weight="19">basic:facility</Category>
</Categories>
<Modifiers member="cafe">
    <Modifier weight="3">vulnerable</Modifier>
    <Modifier weight="2">complete</Modifier>
    <Modifier weight="8">ancillary</Modifier>
    <Modifier weight="214">convenient</Modifier>
    <Modifier weight="297">public</Modifier>
    <Modifier weight="66">private</Modifier>
```

<Modifier weight="4">diverse</Modifier>
<Modifier weight="113">commercial</Modifier>
<Modifier weight="13">cool</Modifier>
<Modifier weight="19">basic</Modifier>
<Modifier weight="97">communal</Modifier>
<Modifier weight="45">social</Modifier>
<Modifier weight="15">busy</Modifier>
<Modifier weight="14">poor</Modifier>
<Modifier weight="33">elaborate</Modifier>
<Modifier weight="15">urban</Modifier>
<Modifier weight="5">orientated</Modifier>
<Modifier weight="3">smaller</Modifier>
<Modifier weight="4">normal</Modifier>
<Modifier weight="46">neutral</Modifier>
<Modifier weight="5">related</Modifier>
<Modifier weight="14">recreational</Modifier>
<Modifier weight="48">remote</Modifier>
<Modifier weight="1">controlled</Modifier>
<Modifier weight="1">necessary</Modifier>
<Modifier weight="11">modern</Modifier>
<Modifier weight="19">stationary</Modifier>
<Modifier weight="7">populated</Modifier>
<Modifier weight="29">local</Modifier>
<Modifier weight="1">offer</Modifier>
<Modifier weight="2">cultural</Modifier>
<Modifier weight="49">physical</Modifier>
<Modifier weight="1">fine</Modifier>
<Modifier weight="5">suitable</Modifier>
<Modifier weight="58">outdoor</Modifier>
<Modifier weight="18">civilian</Modifier>
<Modifier weight="12">nearby</Modifier>
<Modifier weight="3">supporting</Modifier>
<Modifier weight="1">everyday</Modifier>
<Modifier weight="6">small</Modifier>
<Modifier weight="3">advanced</Modifier>
<Modifier weight="2">internal</Modifier>
<Modifier weight="1">limited</Modifier>
<Modifier weight="44">indoor</Modifier>
<Modifier weight="13">tangible</Modifier>
<Modifier weight="13">secure</Modifier>
<Modifier weight="30">key</Modifier>
<Modifier weight="14">large</Modifier>
<Modifier weight="13">major</Modifier>
<Modifier weight="3">strategic</Modifier>
<Modifier weight="12">informal</Modifier>
<Modifier weight="1">inside</Modifier>
<Modifier weight="15">industrial</Modifier>
<Modifier weight="2">awkward</Modifier>
<Modifier weight="10">extra</Modifier>

```
        <Modifier weight="3">outside</Modifier>
        <Modifier weight="41">good</Modifier>
        <Modifier weight="11">common</Modifier>
        <Modifier weight="8">friendly</Modifier>
        <Modifier weight="5">real</Modifier>
        <Modifier weight="10">shared</Modifier>
        <Modifier weight="13">fixed</Modifier>
        <Modifier weight="2">complementary</Modifier>
        <Modifier weight="35">accessible</Modifier>
        <Modifier weight="1">complimentary</Modifier>
        <Modifier weight="23">new</Modifier>
        <Modifier weight="4">popular</Modifier>
        <Modifier weight="13">sufficient</Modifier>
        <Modifier weight="1">intellectual</Modifier>
    </Modifiers>
    <CategoryHeads member="cafe">
        <CategoryHead weight="709">location</CategoryHead>
        <CategoryHead weight="13">stuff</CategoryHead>
        <CategoryHead weight="17">establishment</CategoryHead>
        <CategoryHead weight="559">facility</CategoryHead>
        <CategoryHead weight="5">object</CategoryHead>
        <CategoryHead weight="4">structure</CategoryHead>
        <CategoryHead weight="2">eatery</CategoryHead>
        <CategoryHead weight="15">unit</CategoryHead>
        <CategoryHead weight="1">part</CategoryHead>
        <CategoryHead weight="21">installation</CategoryHead>
        <CategoryHead weight="221">area</CategoryHead>
        <CategoryHead weight="47">building</CategoryHead>
        <CategoryHead weight="1">restaurant</CategoryHead>
        <CategoryHead weight="2">commodity</CategoryHead>
        <CategoryHead weight="36">arena</CategoryHead>
    </CategoryHeads>
</MemberData>
```